INSTITUTE FOR DEFENSE ANALYSES

A Framework for Evidence-Based Licensure of Adaptive Autonomous Systems: Technical Areas

David M. Tate
Rebecca A. Grier
Christopher A. Martin
Franklin L. Moses
David A. Sparrow
James R. Edmonson
Sagar Chaki
David H. Scheidt
Christine D. Piatko
Don Davis
Don Strausberger

March 2016
Approved for public release;
distribution is unlimited.
IDA Paper P-5325
Log: H 16-000680

INSTITUTE FOR DEFENSE ANALYSES
4850 Mark Center Drive
Alexandria, Virginia 22311-1882

The Institute for Defense Analyses is a non-profit corporation that operates three federally funded research and development centers to provide objective analyses of national security issues, particularly those requiring scientific and technical expertise, and conduct related research on other national challenges.

About This Publication

This work was conducted by the Institute for Defense Analyses (IDA) under contract HQ0034-14-D-0001, Project AK-2-3944, "Pedigree-Based Training and Licensure of Autonomous Systems," for the Air Force Research Laboratory (AFRL). The views, opinions, and findings should not be construed as representing the official position of either the Department of Defense or the sponsoring organization.

INSTITUTE FOR DEFENSE ANALYSES

IDA Paper P-5325

A Framework for Evidence-Based Licensure of Adaptive Autonomous Systems: Technical Areas

David M. Tate
Rebecca A. Grier
Christopher A. Martin
Franklin L. Moses
David A. Sparrow
James R. Edmonson
Sagar Chaki
David H. Scheidt
Christine D. Piatko
Don Davis
Don Strausberger

Appendix A.
Introduction

The main body of this paper presents an approach to test, evaluation, validation, and verification (TEVV) of autonomous systems based on licensure.[1] There are a number of areas where existing work must be converted or new techniques developed in order to make this possible. Detailed treatment in the main body would have distracted the reader from the overall approach. Some of these details are presented here, for selected issues:

- Appendix B, Formal Methods—This appendix provides a review of formal methods techniques, their applicability to licensure and autonomy, and extensive references.
- Appendix C, Requirements and Metrics—This appendix describes a process for defining requirements and associated metrics that supports evidence-based licensure (EBL) for autonomous systems.
- Appendix D, Normative Oracle Generation—This appendix describes the attributes of a Normative Oracle that would support EBL for autonomous systems.
- Appendix E, CoActive Design—This appendix describes co-active design, which focuses on the interdependence of the human and the machine performing a joint activity. We might consider this an extension of teaming among humans, but that would imply a high degree of machine sentience.
- Appendix F, Implications of Learning Autonomous Systems for TEVV—This appendix provides a review of formal methods techniques, their applicability to licensure and autonomy, and extensive references.
- Appendix G, Modeling and Simulation Considerations for Licensure of Autonomous Systems—This appendix addresses modeling and simulation's role in licensure of autonomous systems. To do so, key drivers from EBL and autonomy are first identified to form a foundation for analysis.

[1] D. M. Tate, R. A. Grier, C. A. Martin, F. L. Moses, and D. A. Sparrow, "A Framework for Evidence-Based Licensure of Adaptive Autonomous Systems," IDA Paper P-5325, H16-000084 (Alexandria, VA: Institute for Defense Analyses, March 2016). https://www.ida.org/idamedia/Corporate/Files/Publications/IDA_Documents/STD/2016/P-5325.pdf

Appendix B.
Formal Methods

James R. Edmonson and Sagar Chaki

Evidence-Based Licensure: Providing Rigorous Assurance

Autonomous systems will require a high degree of assurance before they can be allowed to make life-critical decisions. Traditional testing-based validation and verification techniques are incomplete and cannot provide the required level of confidence, especially as the target systems reach a level of complexity that far outstrip the capability of current (and future) testing approaches. Formal methods provide a more rigorous approach for assurance but have limitations in terms of scalability and manual effort required. This appendix answers the following questions in the context of formal methods and evidence-based licensure (EBL):

1. What are formal methods? What are the history and state-of-the-art in terms of techniques and tools?
2. What is the relevance of formal methods to autonomy?
3. How can formal methods be used to support EBL?
4. How can formal methods support autonomous system development?
5. How are formal methods uniquely suited to contribute to EBL?

Background on Formal Methods

Formal methods are "mathematically based languages, techniques, and tools for specifying and verifying [complex software systems]" [1]. Unlike empirical testing approaches, formal methods tend to use an exhaustive verification approach that proves or disproves the correctness, safety, or performance of a system using techniques like *formal specification*, *model checking*, and *theorem proving*. The methods differ from empirical testing in that the logic of a program or software product is thoroughly explored, including all possible states, instead of just a subset of scenarios as seen in traditional testing approaches. This attention to the logic provides a powerful means to assess how decisions are made, beyond just what decisions are made, as discussed in the main body.[1]

[1] D. M. Tate, R. A. Grier, C. A. Martin, F. L. Moses, and D. A. Sparrow, "A Framework for Evidence-Based Licensure of Adaptive Autonomous Systems," IDA Paper P-5325, H16-000084 (Alexandria, VA: Institute for Defense Analyses, March 2016).

Classically, correctness of a software product has been specified in the form of preconditions and postconditions, as well as temporal logic formulas. For most developers and engineers, the rigor of specification can be daunting and a barrier to entry for using these more exhaustive formal methods techniques. However, there has been recent work on developing more user-friendly specification mechanisms, such as regular expressions [2] and their variants that do not require a deep knowledge about logic to be used effectively.

Theorem Proving

Computer scientists have studied formal verification of computational systems for decades. This has led to a wide body of research resulting in theoretical concepts and techniques, as well as tools. Initially, the bulk of this work was focused on modeling systems mathematically and then using manual and interactive theorem proving to verify correctness, which was time-intensive, and problems could be inserted during manual steps. A classic example of this technique for software correctness is deductive program verification using the approach of Floyd [3] and Hoare [4]. This work used preconditions (i.e., what was expected before a software function) and postconditions (i.e., what was expected to be true after a software function) as the specification mechanism and was applicable to any programs that always terminated. Though an important result, this work neglected the vast area of programs that ran indefinitely and software that reacted to events like network messages, timer activations, and other asynchronous occurrences in software. In response to some of these blind spots in formal methods practice, Pnueli [5] proposed verifying reactive systems (i.e., those systems that react to events) using temporal logic as the specification mechanism. He developed proof rules for deductive verification of such specifications using theorem proving. In short, he laid the foundation for moving past preconditions and postconditions, which were really only adequate for software that started and stopped predictably, and provided a more lenient specification that modeled the logic (the guts of the software), rather than everything that had to be true before and after execution. Later work has extended this logical expression of software functionality to include preconditions and postconditions combined with constraints for software interaction, networking, timing, security, and other nonfunctional concerns as well. Preconditions for the software map naturally to restrictions on environment or mission necessary to allow licensure.

Model Checking

A major step toward automated verification of systems occurred with the advent of model checking [6] for which Clarke, Emerson, and Sifakis were awarded the 2007 ACM Turing award. In essence, model checking is an algorithm that exhaustively searches the state space of a system (expressed as a Kripke structure, which is essentially a way of describing the transition of state behaviors within software) to decide if it satisfies a

temporal logic specification. This additional expression of state behaviors and transitions has a computation cost for verification. The original model checkers used explicit states and could only verify small systems due to limitations of CPU speed and memory capacity. Subsequent advancements in hardware capability, as well as new innovations in model checking such as symbolic reasoning, abstraction, and symmetry reduction, have enabled the development of state-of-the-art model checkers (such as NuSMV [7] and Murphi) that can handle systems with enormous [8] (even infinite) state spaces. In particular, one of the innovations to better handle scale of states and transitions in software has been bounded model checking [9] that can use propositional satisfiability solvers (a formal methods approach that breaks down software and state features into true-or-false constraints about performance, safety, security, etc.) to find bugs in very complex systems.

Software Verification

Early model checking development and research was driven largely by the need for hardware and device driver verification, rather than systems-of-systems, enterprise applications or autonomous systems. In particular, the infamous Intel Pentium FDIV bug [10] caused a lot of research funding to be diverted to automated verification techniques (including model checking) and led to major developments in formal methods tools for verifying hardware features and performance. More recently, software model checking [11] has emerged as one of the major areas of research and development that focuses on the verification of more complex software. An important milestone is the SLAM project at Microsoft that produced the Static Driver Verifier [12] tool, which is now integrated with the Windows Device Driver Kit. A number of publicly available software model checkers are in active development, as evidenced by the results of ongoing software verification competitions [13]. In addition, the emergence of efficient satisfiability modulo theory (SMT) has led to a resurgence of Hoare-style deductive verification of programs in the form of auto-active verification and associated tools such as Boogie [14] and Frama-C [15]. This automated step to the deductive verification process is important because users of such formal methods tools had previously been required to fully understand the interaction and functionality of a system and transcribe that knowledge into annotations to a verification tool. Instead, auto-active verification techniques allow for software to be analyzed by software tools and for most of the appropriate annotations for constraints, preconditions, and postconditions to be reasoned out without user involvement, which greatly reduces the potential for human error in the verification process.

Concolic Testing

In general, testing suffers from poor coverage of all environments, states, and interactions that a complex system will see during its deployment. This limits its effectiveness, especially for complex systems with large state spaces. However, a recent development, called concolic (concrete-symbolic) testing [16], combines the benefits of

high automation (from testing) with improved coverage (from symbolic analysis). The key idea behind this technique is to use symbolic simulation, instead of classical testing, and gather information from each run (related to branches) to create additional test cases that cover a different part of the state space. Concolic testing has matured to a point where there a number of robust publicly available tools, such as Klee [17], as well as tools used internally by the industry, such as SAGE [18]. A big advantage of concolic testing is that it is directly applicable to large, complex systems, and not only to models constructed from such systems. This means that bugs found by this technique are often real and therefore of high value.

Probabilistic Verification

Another area of formal verification is the analysis of stochastic systems that exhibit randomness and variability. For autonomy, randomness and variability are a big deal, so solutions that address these topics are important. This research has taken two broad paths—probabilistic model checking and statistical model checking. In probabilistic model checking [19], the system to be verified is modeled as a probabilistic automaton (such as a discrete-time Markov chain, continuous-time Markov chain, or a Markov decision process), and the property to be verified is expressed as a formula in a probabilistic temporal logic such as probabilistic computation tree logic (PCTL) [20]. The property is then checked by deriving equations for the model and solving them. These solutions will typically be numerical rather than analytic, which just means that verifiability is often based on a numerical result of a calculation, rather than the analysis of program logic or the internal software functionality. A number of probabilistic model checkers, such as PRISM [21], have been developed and used to analyze a wide range of systems.

Statistical Model Checking

In *statistical model checking* [22, 27], the system under test (SUT) is simulated many times using the Monte Carlo method (i.e., repeated random sampling to obtain numerical results), and each simulation is treated as a random Bernoulli trial (i.e., an experiment with only two possible outcomes: success or failure). The results of these trials are used to estimate the probability of the desired event (i.e., the target property being satisfied) with a target level of precision. In contrast to probabilistic model checking, this technique can be applied to a system directly as long as it can be simulated. However, a large number of simulations are needed to precisely estimate the probability of a rare event. This can be ameliorated to some extent using standard discrete-event simulation techniques, such as conditional Monte Carlo [23], importance sampling [24, 25], and importance splitting [26].

Formal Methods and Their Relevance to Autonomy

Autonomous systems will have to repeatedly make complex decisions during their operation, and these decisions may have far-reaching consequences on human lives. A bad decision could result in financial ruin or physical harm to robots, buildings, or even people. Wherever possible, such decision processes should be exhaustively and thoroughly explored. It is here, in those decision processes where failures are especially problematic, that empirical testing breaks down and formal methods shows true relevance. Empirical testing is limited by the tester's ability to understand systems-of-systems concerns, the impact of adversaries and environments, race conditions (e.g., timing sequences), etc. Formal methods, when applied appropriately, can automatedly detect problems in autonomous systems before they are even deployed.

The major downside of formal methods has traditionally been in the scalability of formal methods techniques to verify complex systems, especially those with high variability and randomness. Some state spaces are simply too big to analyze in a reasonable time. Although autonomous software does push the boundaries of current formal methods techniques, formal methods are the only way to robustly analyze and verify future autonomous decision-making and behaviors. The question is not whether or not formal methods should be applied to autonomous systems, but where to apply formal methods into autonomy.

There are two places to possibly insert formal methods into autonomous system verification: (1) embedded into the decision-making at runtime and (2) applied to the entire software system before deployment of autonomous systems. Given that many of these more complex autonomous systems may have access to large amounts of information and will be forced to make split-second decisions, embedding formal methods checks into the operation of an autonomous system may result in delays of the runtime system that are unmanageable. Thus, for most autonomous systems, we must provide assurance about the correct behavior of autonomous systems statically (i.e., before their deployment). This assurance can be in the form of direct verification and validation of the autonomous system itself, a rigorously assured monitoring and fault-tolerance mechanism, or some combination thereof. Formal methods are well suited to provide this type of assurance.

Use of Formal Methods to Support EBL

As described earlier, there is considerable research into using formal methods to develop software that can be proven to have certain properties. For EBL to be effective, improvements in formal methods, and specifically in techniques to verify the correctness of complex adaptive autonomous systems, are going to be important to identifying issues that are unlikely to be found by traditional testing and validation processes.

Models of computation do exist that are easier to prove and analyze, such as synchronous models of computation between autonomous systems or autonomous system components, which essentially force processing or collaborating elements of the large, complex system to take turns to behave in a more predictable manner. This type of model of computation may be a vital part of making formal verification of such scalable, adaptive autonomous systems tractable and realizable. In addition, abstraction, compositional reasoning, and parametric analysis will be indispensable for verifying complex concurrent systems in a scalable manner. New specification formalisms (e.g., specialized temporal logics or logics to express normative oracles and aids to coactive design) may need to be developed to express desired limits on the behavior of a collection of cognitive agents.

Many bounds on adaptive autonomous systems will be stochastic; for example, we may need to ensure that an autonomous system will stay within a safe specified operating zone with very high probability despite randomness in environment and software subsystems. In addition to new languages for specifying such bounds, advances in probabilistic verification techniques (such as probabilistic model checking and statistical model checking) will be needed to demonstrate that these bounds are achieved. In particular, robust simulation infrastructures for complex autonomous systems will need to be developed to apply statistical model checking. This may tie in with the development of a testbed to facilitate the licensure process. In many situations, ensuring timely behavior by autonomous systems will be critical. In such cases, techniques drawn from the real-time scheduling and real-time queueing theory, as well as new advances in these areas, may be needed to achieve a high level of assurance in a demonstrable manner.

Finally, in cases where static verification is unsuccessful, runtime-assurance techniques (e.g., the Simplex approach and its variants, which may closely resemble techniques identified elsewhere in this report) can provide a strong bound on a system's behavior by monitoring it and switching to safer alternatives when appropriate. However, such techniques must be adapted to a collection of autonomous systems that could collude to avoid detection of malicious behavior. Moreover, this raises the age-old question, "Who will guard the guardians?" Thus, the correctness of the monitoring and switchover logics must themselves be verified through other means. In addition, we must develop architectures (or set of architectural patterns) for autonomous systems that facilitate application of formal and runtime verification in a compositional manner over system components and in an incremental manner over system evolution.

Ultimately, these verification techniques may not suffice to fully answer the question of dependability for the most complex of adaptive autonomous systems. New techniques that enhance existing verification techniques could be needed to handle the scale and complexity of interacting systems in future systems; the synergy between human/agent collaboration; and the inherent uncertainty of operating systems, networking layers, and

real-world events and circumstances between and among thousands of collaborative agents.

Support That Formal Methods Offer to Autonomous System Development

Because formal methods are ultimately based on mathematics, they provide *objective* and *unambiguous* evidence of correctness that can be shared and examined independently by several licensing and certification regimes. There is increasing realization that autonomous systems will, at least in part, have stochastic and unpredictable behavior. This will emerge from at least two sources. First, the systems will operate in uncertain environments and deal with random external inputs. Second, some will internally use sophisticated techniques, such as machine learning, that provide high capability at the cost of unpredictable behavior. Formal methods are also suited to provide assurance about such stochastic systems, since there is a well-established theory and practice of analyzing such systems based on probability and statistics. The development of autonomous systems will also require a systematic approach if we are to achieve a minimal level of assurance. Given the high cost of failure of such systems, they cannot be developed solely in a produce-cycle and feature-driven manner. There is now considerable evidence that the use of formal techniques earlier in the life cycle of a system (e.g., at the design phase) leads to much lower production cost and to systems that have high quality, modifiability, and maintainability. These lessons will likely carry over to autonomous systems as well. Finally, formal methods can be incorporated into automated tools and can therefore be applied to autonomous systems during their entire life cycle in a routine manner.

Unique Characteristics of Formal Methods for Contributing to Evidence-Based Licensure

An EBL process will need new techniques to address verification problems with machine learning in current and next-generation autonomous systems. A continuously learning system, where an autonomous agent becomes smarter and more capable over time, is unlikely to exhibit the same performance characteristics and limitations that it did when first certified. Consequently, for most learning systems, the adaptive autonomous system will need a continuous-certification process for safety, adaptability, and coherence. Formal methods provide a promising basis for developing such systems.

We may be able to develop a verification methodology for continuous learning systems via an abstraction process for the autonomous software system that converts the entire learned system into a form that is more readily analyzable by formal methods processes, such as a finite-state machines with currently learned values, behaviors, and automated procedures to verify the finite-state machine against known operating environments. Such abstraction techniques are not currently available, and research,

processes, and tools would undoubtedly be helpful for gaining confidence in existing and future learning-based systems, especially in arbitrary, unknown operating environments.

In addition, suitable architectures could help us decompose the system into more manageable components with well-defined interfaces where appropriate formal verification techniques (e.g., software-model checking or statistical-model checking) could be applied. At the very least, such techniques and architectures would allow licensure specialists to better understand the limits of an autonomous system and when and where the system should be used and how it should definitely not be used, long before the system is placed in a position that could violate the safety or correct operation of human operators, collaborators, and other equipment. Integration of diverse formal methods in the context of a single system development is also a fertile area for further research.

Conclusion

In summary, research in formal verification over the last several decades has produced a range of formalisms, techniques, and tools that have been applied to systems ranging from device drivers to distributed software. Formal methods, when successful, provide a high degree of assurance about a system's behavior (since they are exhaustive) and yield objective evidence (in the form of proofs and counterexamples) to support its results. Moreover, formal methods can be applied statically at design and development time, before actual system deployment. This is invaluable in detecting failures that lead to catastrophic results, and formal methods have the potential to provide hard facts about how a system might operate in a real-world environment before it is actively deployed. However, formally verifying systems is time intensive and in many cases intractable, even for small, non-distributed autonomous systems. Scaling to hundreds or thousands of autonomous agents or systems makes the problem even harder.

References

[1] Edmund M. Clarke and Jeannette M. Wing. "Formal Methods: State of the Art and Future Directions." *ACM Comput. Surv.* 28, 4 (December 1996): 626–43.

[2] James C. Corbett, Matthew B. Dwyer, John Hatcliff, Robby. "Expressing Checkable Properties of Dynamic Systems: The Bandera Specification Language." *STTT* 4 (1) (2002): 34–56.

[3] R. W. Floyd. "Assigning Meanings to Programs." *Proceedings of the American Mathematical Society Symposia on Applied Mathematics* 19 (1967): 19–31.

[4] C. A. R. Hoare. "An Axiomatic Basis for Computer Programming." *Commun. ACM* 12 (10) (1969): 576–80.

[5] Amir Pnueli. "The Temporal Logic of Programs." *FOCS*, 1977: 46–57.

[6] Edmund M. Clarke and E. Allen Emerson. “Design and Synthesis of Synchronization Skeletons Using Branching-Time Temporal Logic.” *Logic of Programs,* 1981: 52–71.

[7] NuSMV Model Checker. http://nusmv.fbk.eu/.

[8] Jerry R. Burch, Edmund M. Clarke, Kenneth L. McMillan, David L. Dill, and L. J. Hwang. “Symbolic Model Checking: 10^20 States and Beyond.” *Inf. Comput.* 98 (2) (1992): 142–70.

[9] Armin Biere, Alessandro Cimatti, Edmund M. Clarke, and Yunshan Zhu. “Symbolic Model Checking without BDDs.” *TACAS,* 1999: 193–207.

[10] https://en.wikipedia.org/wiki/Pentium_FDIV_bug.

[11] Ranjit Jhala and Rupak Majumdar. “Software Model Checking.” *ACM Comput. Surv.* 41(4) (2009).

[12] Thomas Ball, Ella Bounimova, Vladimir Levin, Rahul Kumar, and Jakob Lichtenberg. “The Static Driver Verifier Research Platform.” *CAV,* 2010: 119–22

[13] http://sv-comp.sosy-lab.org/2016/.

[14] http://research.microsoft.com/en-us/projects/boogie/.

[15] http://frama-c.com/.

[16] Koushik Sen. “Concolic Testing.” *ASE* 2007: 571–72.

[17] KLEE tool. https://klee.github.io/.

[18] Patrice Godefroid, Michael Y. Levin, and David A. Molnar. “SAGE: Whitebox Fuzzing for Security Testing.” *Commun. ACM* 55 (3) (2012): 40–44.

[19] Marta Z. Kwiatkowska and David Parker. “Advances in Probabilistic Model Checking.” *Software Safety and Security,* 2012: 126–51.

[20] Hans Hansson and Bengt Jonsson. “A Logic for Reasoning about Time and Reliability.” *Formal Asp. Comput.* 6 (5) (1994): 512–35.

[21] http://www.prismmodelchecker.org/.

[22] Håkan L. S. Younes. “Ymer: A Statistical Model Checker.” *CAV,* 2005: 429–33.

[23] Joshua C. C. Chan and Dirk P. Kroese. “Rare Event Probability Estimation with Conditional Monte Carlo.” *Annals of Operations Research* 189 (2011): 43–61.

[24] Edmund M. Clarke and Paolo Zuliani. “Statistical Model Checking for Cyber-Physical Systems.” *ATVA,* 2011: 1–12.

[25] Jeffery P. Hansen, Lutz Wrage, Sagar Chaki, Dionisio de Niz, and Mark H. Klein. “Semantic Importance Sampling for Statistical Model Checking.” *TACAS,* 2015: 241–55.

[26] Cyrille Jégourel, Axel Legay, and Sean Sedwards. “Importance Splitting for Statistical Model Checking Rare Properties.” *CAV,* 2013: 576–91.

[27] David Kyle, Jeffery P. Hansen, and Sagar Chaki. “Statistical Model Checking of Distributed Adaptive Real-Time Software.” *RV*, 2015: 269–74.

Appendix C.
Requirements and Metrics
David H. Scheidt and Christine D. Piatko

Scope

We describe here an overarching process to define autonomous system requirements to support evidence-based certification and licensure, which will develop and maintain confidence in system dependability throughout the system life cycle (TEVV 2015). Note that, interpreted broadly, autonomous system requirements include base requirements for use cases of all systems that could possibly be made autonomous. All forms of unmanned vehicles, as well as immobots such as electrical grids, fluid-distribution systems, and cyber-physical systems, have the potential to be made autonomous. Accordingly, general autonomous systems requirements include requirements of all autonomous uses of vehicles and immobots. We do not fully enumerate specific requirements related to the direct testing and licensing of any single particular autonomous system or subcomponent. Rather than attempt to address requirements for the systems in toto, this appendix focuses on addressing the derivation of *requirements for the evidence-based licensure of the autonomous system*. We describe the overall conceptual process by which autonomy requirements and metrics for licensure can be elicited.

As described in the main body of this report, the goal of licensure is to provide assurances that the autonomous system will perform well under the conditions for which it is being licensed.[1] Licensing the system as a whole involves building up a constructive assurance argument from evaluations of the system in relevant circumstances and from evaluations of system components. The system includes a physical plant, system software, and decision algorithms. The physical plant consists of all mechanical, electrical, and potentially chemical components within the systems that allow it to perform the assigned tasks, including motors, drivetrains, actuators, sensors, processors, and network devices. The systems software interprets signals from system sensors and, in accordance with the decision algorithms, controls actuators, internal processes, and communications with the external world. Note that tele-operated unmanned vehicles and immobots are composed of equivalent physical plants and system software.

[1] D. M. Tate, R. A. Grier, C. A. Martin, F. L. Moses, and D. A. Sparrow, "A Framework for Evidence-Based Licensure of Adaptive Autonomous Systems," IDA Paper P-5325, H16-000084 (Alexandria, VA: Institute for Defense Analyses, March 2016).

Tele-operated systems software is traceable to well-formed system specifications that can be used as a basis of a formal test plans. Unlike tele-operated systems, autonomous systems are expected to interact with the external world and devise a course of action that may not have been explicitly defined within the specifications. The task of determining the course of action lies with the decision algorithms embodied within the autonomous system. Unlike tele-operated systems, which explicitly define "what" the desired response to an external stimulus is, autonomous system specifications define "how" the desired response should to be produced. Accordingly, validating that system software correctly satisfies system specifications is insufficient evidence for licensure; system validation requires licensure of both the algorithm and its implementation. For example, proving that a robot's path-planning software correctly defines the path-planning algorithm specified in the design is insufficient evidence for depending on the robot to correctly find a path; testing the robot requires that the robot test team validate that not only was the path-planning software correctly coded but that the algorithm that was encoded will, under all conditions for its licensure, satisfy the system specifications. Further, the system's physical plant, software, and decision algorithms must all be validated under appropriate conditions for their suitability, as well as integrate available assurances for the system as a whole tested under a variety of conditions.

As discussed in the main body of the report, there are several approaches to the licensure of an autonomous system and its components. One approach is via formal mathematical proofs that the system will work as described. Section 2 provides a discussion of formal methods along these lines. Note that formal methods can be used to validate the physical plant, the system software, and the decision algorithms.

A second approach is the use of controlled experiments, putting the system under test in controlled, sanitized settings that provide fixed assurances of suitable performance in these specific conditions. This is analogous to what is often done to test subunits of larger physical systems. Unit testing of physical components contained within an autonomous system can be used to characterize the requirements and performance of specific sensors, drivetrains, software models, and operational elements. Controlled experiments for autonomous systems are more challenging than testing of equivalent tele-operated systems because the subsystems within an autonomous system are expected to observe the actions of "thinking actors" in the outside world and devise a course of action that will, in turn, generate actions on behalf of the external actors. The feedback between autonomous system decisions and decisions of outside actors explodes the state space to make test plans composed entirely of controlled experiments infeasible for all but the simplest of autonomous systems.

The third approach to licensure, and the one emphasized throughout this report, is evidence-based licensure. In contrast to formal methods or controlled experimental approaches, such licensure will necessitate observing the system's performance in

conditions appropriate to its licensure—including in native environments ("in the wild"). The goal of a composable licensure approach will be to attempt to avoid going through lengthy, expensive, full system test processes again and again, instead building an argument for licensure related to how the system is expected to perform based on its components and related system tests and licenses.

The major challenge for developing licensure requirements in support of such composable, evidence-based licensure will be eliciting the many possible inter-dependencies between the licensure of individual system components and the overall system, as well as describing how each relates when tested under different sets of conditions. We describe here a method of eliciting licensure requirements and cross-requirement dependencies in order to develop a set of licensure life-cycle requirements matrices.

Further research will be necessary to fully develop this approach into an accepted licensure requirements-generation process. New techniques will be required for several aspects:

- Articulating adaptability licensure requirements and how requirements might be transient over time.
- Describing cross-requirement dependencies.
- Determining when and how component suitability can be composed to grant broader system licensure.

Research and development will also be necessary to begin to integrate this type of requirements-generation process across the broad range of system elements in preparation for evidence-based licensure test and evaluation.

Licensure Requirements and Metrics

Licensure will be driven by assurances of suitability for the autonomous system to perform well in its environment. Thus, identifying licensure requirements and identifying corresponding metrics—including measures of performance (MOP), measures of effectiveness (MOE), and suitability—will be intertwined and, in some sense, almost two sides of the same coin. Test, evaluation, validation, and verification (TEVV) using concrete metrics will be necessary to provide the assurances that the system demonstrates the dependability necessary to support licensure.

Licensure and Representative Autonomous Systems Examples

Teen Driver Licensing

An analogy can be drawn between licensure and a teenager's getting a learner's permit, then a driver's license, and then driving privileges from parents for use of the family car. Some aspects of driving license tests are done under controlled conditions (e.g., multiple choice tests to confirm knowing the "rules of the road"), whereas others are evaluated with both controlled and "in the wild" performance tests, where actual driving behavior is witnessed and graded by an observer, with specific subtests such as parallel parking.

Initial licensure often comes with suitability constraints. For example in Maryland, a new driver "may not drive with passengers under the age of 18, other than immediate family members, for the first 151 days, without a qualified supervising driver."[2] And such a driver "may not drive between 12 midnight and 5 AM unless: a supervised, licensed driver who is at least 21 years old and has 3 years of driving is with them" or if they are "driving to or from a job, official school activity, organized volunteer program or are participating in an athletic event or related training session." The driver has an 18-month provisional period, which is reset if there are any accidents or other driving infractions during the period; accumulating severe enough infractions can result in loss of licensure.

The American Automobile Association further recommends that parents only gradually expand a teenager's driving privileges as he or she demonstrates competence, analogous to expanding the scope of licensure of the system being suitable under increasingly complex conditions.[3] The recommendation is for quarterly to half-year checkpoints before allowing a newly licensed teen to use a family car with more privileges. Expanded driving privileges can include a progression such as the following: Is the teen allowed to drive at nighttime or not? How late (sundown, 9 p.m., 10 p.m., 11 p.m., midnight)? Can the teen have other teen passengers in the car in the daytime (none, one, sometimes one or two, sometimes several) or similarly at night? Can the teen drive only when it is dry during the day or also while it is raining (if so, in light rain, moderate or heavy), and similarly at night? What types of roads is the teen allowed to drive during the day (local, all but highways, most types), and similarly at night?[3]

(Note parental assurances may also require the teen to demonstrate dependability with additional aspects of car usage, outside of licensure, such as returning the car at agreed-upon curfew times, not letting other teens drive the vehicle, etc.)

---

[2] All quotes in this paragraph are from http://www.mva.maryland.gov/drivers/rookie-driver/general-provisional.htm.

[3] http://teendriving.aaa.com/wp-content/uploads/2015/01/Parent.Teen_.Driving.Agreement.pdf.

In this example, gradual permission for additional licensure and extended autonomous driving privileges under suitable conditions involves initial formal testing, but then expands based on checkpoint evaluation of successful performance through a series of gradually more challenging conditions.

This licensing and checkpoints recommendation structure for expanding privileges has been developed over a long period of time, through observation of many teenage drivers.

The challenge for licensure requirements generation for autonomous systems will be to develop similar checkpoint strategies in an effective manner for a wide variety of autonomous systems.

Representative Autonomous Systems

When considering autonomy licensure requirements, we assume that autonomous systems adhere to the general-purpose architecture shown in Figure C-1 in which a system contains a reasoning engine that produces decisions. The reasoning engine is implemented in software that is part of a larger body of software used to control the system under test. Decisions are based on observations produced by messages from off-board sensors, operators and peer systems and observations from the system's own sensors. The reasoning engine produces decisions that are executed by the system, producing actions that in turn produce desired effects in the operating environment.

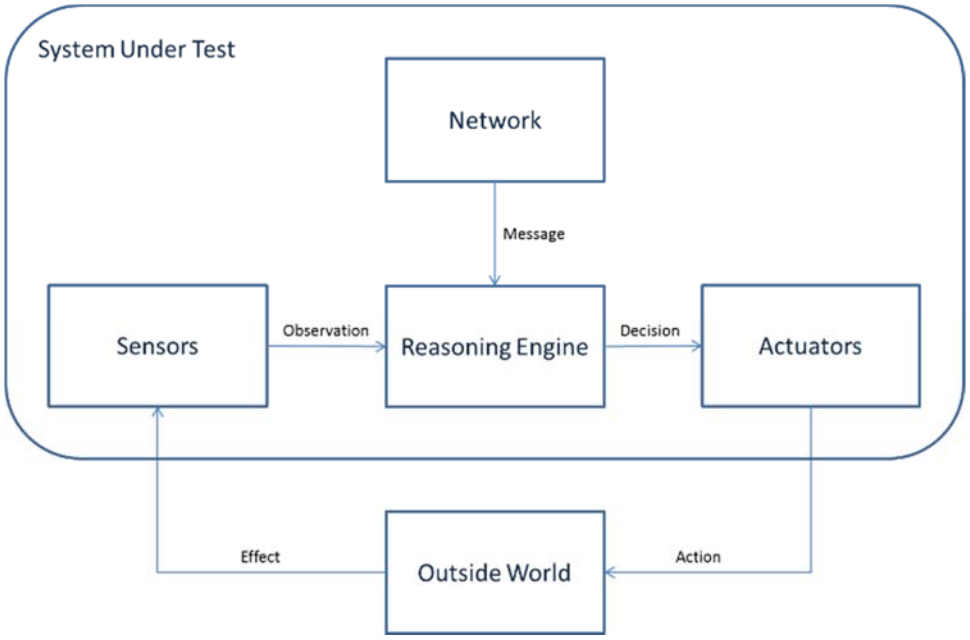


Figure C-1. We assume a general-purpose autonomous system architecture in which a reasoning engine produces decisions based upon observations and messages.

We give here an example of an autonomous system to be used throughout this appendix. Consider an autonomous tractor-trailer, the natural next evolution of the self-driving cars under development today. The tractor-trailer must be able to drive safely across long distances, in a timely fashion, to deliver goods between locations, while obeying the "rules of the road," avoiding collisions, etc. Much of the physical plant for our notional tractor-trailer is identical to a modern truck and its attendant intermodal trailer. To support the autonomous nature of the tractor-trailer we augment the base physical plant by adding some sensors, including LIDAR (light detection and ranging); cameras and microphones; additional computational infrastructure to host the additional systems software; and a radio for communications to the company's dispatcher and police. We assume the tractor-trailer includes system software for device-level control of the power train, drivetrain, and braking system (i.e., we assume it already has an anti-lock braking system). The autonomous system includes additional software modules that implement the system's operational elements, which include algorithms capable of detecting and tracking obstacles, other vehicles, pedestrians, and the occasional wayward animal; collision-avoidance protocols; goal-directed behaviors that enable the tractor-trailer to adhere to traffic laws; localization software that provides tractor and trailer location and pose at all times; and a path planner that produces a course to the dispatcher-provided destination given current traffic patterns and blockages.

A Process for Defining Licensure Requirements

A six-step process for eliciting licensure requirements is shown in Figure C-2. (The sample set of arrows is incomplete because, in practice, this would be a densely filled matrix.) This process can be used to develop a comprehensive requirements set that can be used as a licensure basis through the system life-cycle including develop, test, and post-delivery.

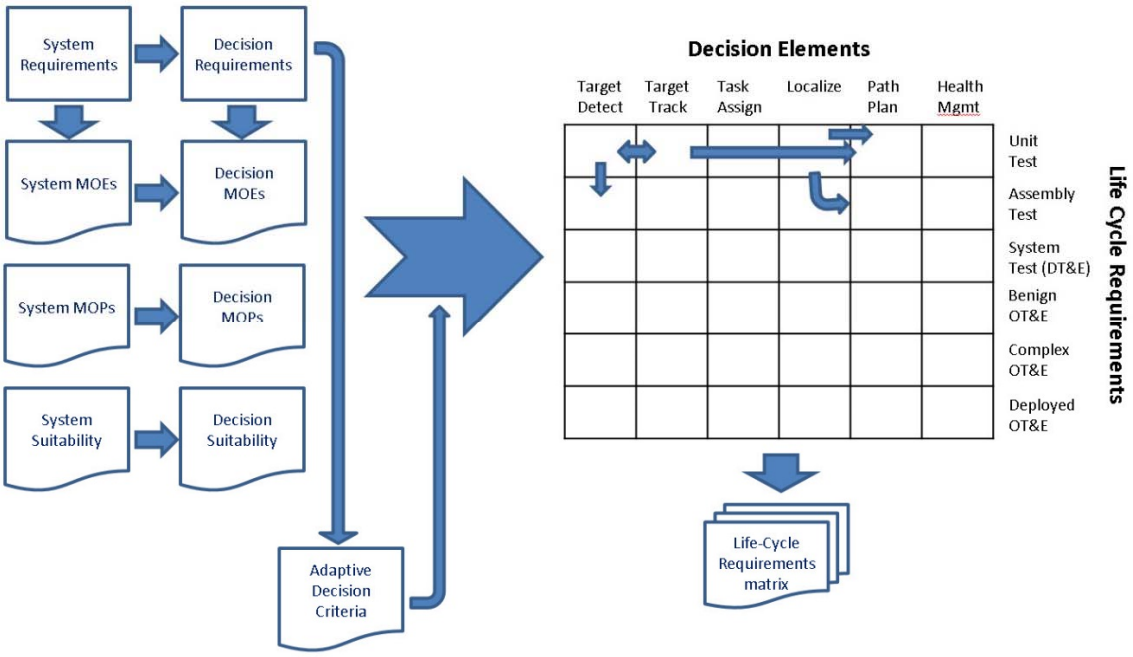


Figure C-2. A Conceptual Process for Eliciting and Articulating Requirements for Autonomy Licensure

Step 1: System Specifications

The first step in defining licensure requirements is determining *system specifications*. This step will lay out the licensure requirements for the autonomous system as a whole. This step focuses on system performance overall, which includes decision-making and non-decision-making aspects of the system.

The associated licensure for these system specification requirements will require corresponding system measures: system measures of performance (MOPs), system measures of effectiveness (MOEs), and evaluations of system suitability. Here, suitability will be broken down into licensing steps in a graduated process (e.g., for simple environments, then increasingly difficult environs). MOPs, MOEs, and suitability descriptions are combined into a systems specifications document, a partial example of which is shown in Figure C-3.

System Specifications

Representative System Specification Requirements for Autonomous Tractor Trailer	***Associated Metrics***
Measures of Performance	
"stops at red lights"	Brakes successfully activate to stop vehicle within *n* meters of red light or associated striping
"follows local traffic laws."	
Measures of Effectiveness	t_0 = start time t_p = predicted arrival time t_a = actual arrival time
Arrives "on-time"	*on-time metric* = $\Delta(t_p - t_a)$
Arrives "as fast as possible"	*as_fast_as_possible* = min t_a (t_a- t_0)
Suitability Measures	
May be operated on federal and state roads (not certified for off-road use, for example this platform may rely on standardized signage).	
May be operated within terminals.	
Can only operate if no flat tires.	50 <= metric psi of each tire <= 60
…	…

Figure C-3. The first product of the autonomy requirements process is a detailed specification that defines requirements for the entire system, which include the physical plant as well as the autonomous decision-making apparatus.

Why do we need metrics for licensure of autonomous systems? The point of measuring autonomy will be to help assess whether or not the military commander can depend on the overall autonomous system to do its job. For each autonomous system to be accepted and deployed, we will need predictive capabilities that provide appropriate levels of dependability and licensure so that the autonomous system will be capable of accomplishing its mission. The results of autonomous system measurements should satisfy licensure requirements and exhibit dependability.

Note here we are interested in measurements that will lead to dependability of mission effectiveness of autonomous *systems* vs. autonomy alone. Notable attempts to define and measure levels of *autonomy* have been made by NIST (e.g., Huang 2008) and AFRL (e.g., Clough 2002). However, as noted by the Defense Science Board (2012), autonomy measurements in and of themselves are not particularly useful in assessing the operational utility of autonomous *systems*. A simple autonomous system may not have much "intelligence"—that is, it may in some sense have "stupid" autonomy—but it may be able to do its mission very well. Measuring, for example, the "IQ" of an autonomous system may not be that useful, as the point of measuring autonomy is trying to measure whether the system will do a job for you dependably. There may be situations where the system has

a high level of autonomy—but should not be "trusted" (cf. the behavior of the autonomous computer system HAL in Stanley Kubrick's *2001: A Space Odyssey*).

Autonomy measurements relate closely to command-and-control (C2) measurements. After all, autonomy can be viewed as a form of C2 in which a machine is authorized to make a decision. In the influential book *Power to the Edge* (Alberts and Hayes 2003), the OSD Command and Control Research Program (CCRP) makes a compelling argument that mission effectiveness can be approved when "edge warfighters" are empowered to make decisions. When the edge warfighter is an unmanned vehicle, Alberts and Hayes's argument to empower the edge can be viewed as an argument for autonomy. In Kass (2006) measures of performance (MOPs) and measures of effectiveness (MOEs) are used to measure C2, using the logic that autonomy is a form of C2. We follow their concepts here.

In this effort, we follow the recommendations made by the OSD Autonomy Community of Interest's Test, Evaluation, Verification and Validation working group Technology Investment Strategy 2015–2018 (TEVV 2015). The TEVV report recommends the use of standardized operational MOPs for measuring autonomous systems. We also recommend the use of MOEs.

Measures of Performance

MOPs address *whether the autonomous system delivers the features it is intended to have*. Such metrics can be viewed as constraints with pass/fail criteria. In a development context, MOPs are things like speed and payload.

In a mission-assessment context, an MOP for the autonomous tractor-trailer may be if it reaches its destination. For a package-delivery drone unmanned aerial vehicle (UAV), one measure of its performance may be a binary variable, that is, whether it flies and makes its way to its destination goal. Similarly, for a swarm of UAVs, it either succeeds in mapping its responsible area or it does not.

Whether an autonomous system meets legal and ethical constraints can also be a measure of performance (Gillespie and West 2010).

Measures of Effectiveness

MOEs measures *how completely the system can accomplish its missions*, or degrees of quality in how well the system accomplished its mission. This is not done in isolation, but over the context of all possible circumstances. It is a graduated measurement, measuring the quality of how well the system does something in a particular context.

For example, searching by a random sweep will help a package delivery drone find its goal and achieve a good MOP, but it could be expensive in terms of time taken, leading to late deliveries, and thus a low MOEs. Using an intelligent search algorithm, or a path planner, would likely have a much higher MOE associated with timely deliveries.

Such measures must be considered in context of what the system has been asked to do and what kind of decisions it has been asked to make.

Suitability

Suitability measures should be used to *characterize the conditions under which the autonomy works*. Referring to Figure C-1, an autonomous system's decisions can be defined directly as a function of observations and messages and indirectly as a function of the operating environment in the outside world. Suitability measures can be used to correlate measures of performance and effectiveness to both these types of operating conditions.

Characterizing the External Environment at the Time of the Decision

The physical environment of the system will play a role in its ability to operate, as well as its ability to operate effectively and efficiently. For example, the system may only function well when certain guidelines are met; for example, if it is dark out, it will work, whereas if it is light out, it might not work well, or at all. The MOPs and MOEs should be a function of the operating environment of the system under test described by such suitability characteristics. The MOPs should be correlated to parameterized operating conditions as to whether the system can satisfy its objectives. The MOEs can then also be used to add more information on effectiveness aspects, such as how long the system is going to take to accomplish its mission, as a function of the range of suitable operating conditions.

For example, the tractor-trailer may not be licensed to operate in snow. A package-delivery drone may not be licensed to operate during tornado watch or warning conditions, or more generally if the wind is above a certain speed. A swarm of UAVs may be licensed to operate at night, but with expected, characterized degradations in its surveillance capabilities.

Note that additional, non-physical external factors can also play a role in determining the suitability for a decision in the operating environment. For example, a decision may be suitable only if it is not against policy (Alberts and Hayes 2003).

Characterizing the Internal Environment at the Time of the Decision

Suitability measures of the internal environment can be used to help characterize what needs to be known at the time of the decision. For example, the system may require a certain percentage of sensors to be operating to feed input to the reasoning engine. Such suitability metrics can again be used to parameterize the MOPs and MOEs, given the appropriate range of operating conditions for the system. As an example, a swarm of UAVs may know it will not operate well if less than 50% of the swarm sensors are operational.

Characterizing the Results (meta-metrics) of the MOEs

Finally, it will sometimes also be desirable to measure "meta-metrics" of the MOEs, to characterize the system over the variety of potential suitable conditions. For example, cohesion could be one measure of performance over a range of potential operating conditions. The potential state space the autonomous system can perform in—that is, the areas it satisfies mission requirements—might be cohesive in some way (e.g., relatively stable over a continuous interval of temperatures or wind speeds). On the other hand, if it is not cohesive it may be difficult to predict when the system just does not work—perhaps statistically 90% of time it works, 10% with no rhyme or reason it just does not. In that case, it might be argued that overall system performance is not cohesive. A less cohesive system might be considered less dependable.

Step 2: Decision Requirements

This next step of eliciting requirements for licensure focuses *on the decisions that need to be made* by the autonomous system. For example, to deliver a package, a package-delivery drone may require not just effective sensing and navigation but also effective decision methods to position its sensors to know when it has reached its destination.

Given the input of information from sensors and other subsystems, an autonomous system will be producing other information or produce control decisions. We focus here on the decision itself, not the execution of decision. This may also include such aspects as when the system recognizes that something in the system itself is broken and decides to fix it.

Such decisions can be framed as triples—given this type of data, the system produces this decision, suitable for this particular environment. Note that these decision requirements for licensure will then also need to be framed in terms of MOEs, MOPs, and suitability assessments, drawing in suitability arguments that appropriate decisions are being made in appropriate environments. Decision requirements are defined in a *Specifications* document, a partial example of which is shown in Figure C-4.

Decision Specifications

Representative Decision Requirements for Autonomous Tractor Trailer	*Associated Metrics*
Must identify suitable route	
Must decide to stop at stop signs	Makes the decision to stop at the stop sign
Must decide to continue on from stop sign iff way is clear	
Must decide to continue on from stop sign iff earlier stopped traffic has continued	
Must decide to pull over if tire is flat and suitable pull-over location	
Must learn where likely traffic tie-ups are	

Figure C-4. The second product of the autonomy requirements process is the development of the decision specifications that separate out those requirements

Step 3: Adaptability Decomposition

Autonomous systems are, by their very nature, adaptive systems in that they are required to observe the world in which they operate and adapt to changes in the world. Step three in defining licensure requirements is to define adaptability requirements for the system. The next key consideration for licensure requirements will be *when and how they should evolve as the autonomous system adapts*. To explore this concept we introduce three levels of adaptation—(1) adaptation to static uncertainty, (2) adaptation to dynamic uncertainty, and (3) adaptation to evolutionary uncertainty:

- *Adaptation to static uncertainty*—Autonomous systems that must adapt to static uncertainty are required to devise and execute a course of action when confronted with unexpected circumstances; however, the world in which the autonomous system operates does not itself adapt to the autonomous system.
- *Adaptation to dynamic uncertainty*—Autonomous systems that must adapt to dynamic uncertainty are required to manage interactions between outside decision-makers, which may or may not be adversarial, and decisions they may make in response to actions made by the autonomous system; however, the behaviors made by outside decision-makers are static, and the autonomous system can expect decision-makers to act in a predictable manner.
- *Adaptation to evolutionary uncertainty*—Autonomous systems that must adapt to evolutionary uncertainty are required to manage interactions with outside decision-makers that are capable of changing their behaviors over time, including an ability to appropriately respond to adversaries that are capable of learning.

The level of adaptation of the decision algorithm can substantially affect the evidence needed for licensure because initial requirements for the system to respond in particular ways in particular conditions may not last (i.e., the licensure requirements may be *transient*). An adaptability decomposition will need to express the permanence of the associated adaptability licensure requirements (i.e., whether or not certain requirements are *transient*). Associated licensure requirements will also be necessary for forms of meta control, which change the way the system makes decisions to match changing environments. The lifetime of such decisions will need to be represented. System adaptability requirements are defined in an *Adaptability Decomposition Matrix*, a partial example of which is shown in Figure C-5.

Adaptability Decomposition Matrix

Autonomous Tractor Trailer Adaptability Decomposition	**Static Uncertainty**	**Dynamic Uncertainty**	**Evolutionary Uncertainty**
ID Suitable Route	Localization Error Road Closures Road Conditions	Traffic (other drivers)	Traffic patterns
Stop at Stop Signs	Lighting Conditions (ability to sense) Damage to Signs	Other drivers	Rome drivers ≠ Columbus drivers
Pull over when Tire Flat	Tire Pressure Localization Error Lighting Conditions (ability to sense)	None	Pirellis last longer
…	…	…	…

Figure C-5. The third product of an autonomous system requirements analysis is the adaptability matrix, which identifies change that must be managed by the autonomous system.

Framing adaptability of licensure requirements will be one of the most challenging aspects for requirements licensure generation. It will require much further research on how to best describe and formulate such adaptability licensure requirements specifications.

Step 4: Operational Decomposition

Next, the system requirements for licensure will need to be decomposed according to the different types of operational elements of the autonomous system (e.g., sensory, control, collaboration, etc.). Again, for each element, appropriate licensure requirements measures will need to be derived. One can develop MOPs, MOEs, and suitability measures for operational elements just as for the system. Again, for each operational element, suitability will need to be broken down into graduated licensing steps. (See vertical axis of

Cognitive Elements on the right of Figure C-2.) Operational decomposition is defined in a matrix of the same name, a partial example of which is shown in Figure C-6.

Step 5: Test and Evaluation Decomposition

Next, the system will need to be decomposed along its test and evaluation (T&E) elements: unit tests, assembly tests, field tests (developmental T&E), and operational T&E, in a progression of environments. Torens and Adolph (2014) describes a six-step process for the test and evaluation of autonomous unmanned air systems that may be used as a model for TEVV of all forms of autonomous systems (Figure C-7). Recognizing that autonomous systems that are capable of learning will modify their autonomous behavior post deployment we extend Torens and Adolph, TEVV process to include a seventh step, post-deployment testing. During our test and evaluation decomposition step, we map each step in our "Torens plus one" process to requirements producing during the operational decomposition step which produces the two-dimensional requirements matrix shown in Figure C-2. Note each element is license-based.

Detailed component requirements should be derived from the system-level autonomy requirements and mapped into component-level metrics to form the basis of unit and component testing, which can then be used to help form a constructive validation and licensure of the overall system.

Operational Decomposition Matrix

Autonomous Tractor Trailer Cognitive Decomposition	**Localization**	**Path Planning**	**Path Assessment**	**Object Detection**	**Object Classification**	**Object Tracking**	**Fault Management Diagnosis**
Must ID suitable route	Localization Orientation	Select route along allowed roads	Assess timing of route	Learn of historical blockages			
Must reroute when necessary	Localization Orientation	Select route along allowed roads	Assess timing of route	Accept communicated blockages			
Must stop at stop signs	Localize self Localize sign Localize stop stripe	Route to location at strips, or behind car in front		- Cars (in front) - Cars @ other signs - Cars or roads not impeded by sign (e.g. on primary cross-road) - Stop signs(s) - Stripe	Cars	Cars - Cars	
Decide to pull over if tire flat	Localize self	Route to safe pullover		Shoulder Objects on shoulder	Cars	Cars	Tire failure
…	…	…	…	…	…	…	…

Figure C-6. The fifth product of the autonomy requirements process is the operational decomposition, which breaks down decision-making requirements into sub-requirements for each component within the architecture.

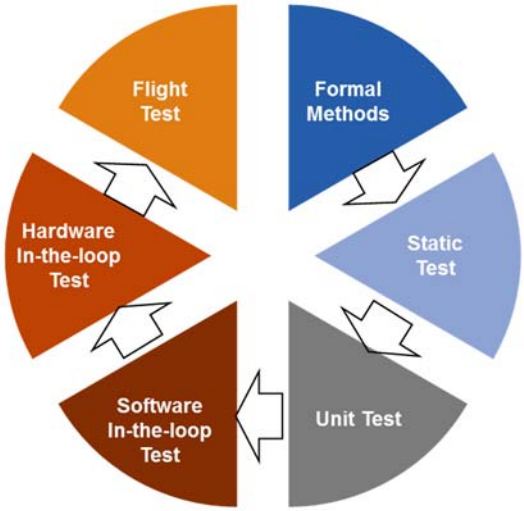


Figure C-7. Torens and Adolph (2014) envision a six step process for TEVV of autonomous systems that begins with formal analysis of autonomy algorithms and ends with system testing.

Evidence-based Assurance Plan

Autonomous Tractor Trailer Test and Evaluation Decomposition	**Formal Methods**	**Static Test**	**Unit Test**	**Software in the loop**	**Hardware in the loop**	**Flight Test**
Car detection			Google car ride-along testing			
Localization			Assess timing of route			GPS evidence
Path Planning	Theorem proving to show no deadlock	Hoare logic proof		-		
Tire pressure						UL tested
Stop sign detection				Emulation testing	Lab testing	Cars
…	…	…	…	…	…	…

Figure C-8. After defining the requirements in detail, an Evidence-based Assurance Plan defines the assurance methods that will be used to determine dependability of each component. Note that diverse methods may be used to provide the assurance necessary to grant a license.

Step 6: Mapping Dependencies

The next step in the licensure requirements process will be mapping dependencies between the T&E licensure elements for the various system elements.

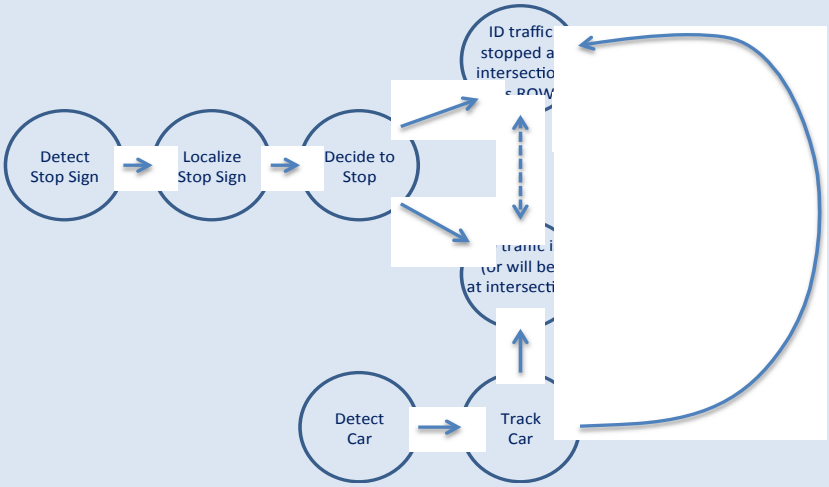
These cross-requirement dependencies will be key for enabling licensure composability. See some example dependencies depicted in the life-cycle requirements matrix in Figure C-2. (The sample set of arrows is quite incomplete; in practice, we would expect this to be a densely filled matrix.) The documents defined in earlier steps of the process build up to an evidence-based assurance plan, which provides a comprehensive plan for evidence-based licensure of the autonomous system. Figure C-8 shows a notional version of such a plan.

Mapping dependencies will require developing methods to ensure that descriptions and licensure requirements are complete enough that they allow composition of assurances for granting licensure under suitable conditions. Note that dependencies may have mismatches that might need to be resolved. The licensure tests between components may have been done at very disparate levels of detail, or have different forms of construction. As an example, consider airspace deconfliction for a package-delivery drone. It may use a licensed localization algorithm, as well as a licensed path-planning algorithm that does well flying while following major roads and that does not let it collide with static obstacles such as trees or wires. Each of these components may have been tested separately. Collectively, however, these components may or may not be at the right level of fidelity to be able to compose them and ensure that the UAV will not hit a car coming along the road. Additional licensure requirements may be necessary, such as licensure requirements related to the performance, effectiveness and suitability for the UAV to detect and avoid a moving car. Or the map may have licensure to include frequent, dynamic updates of changing obstacles (such as a car moving), and the UAV have licensure to navigate a map that has dynamic updates at a certain rate. Similarly, an autonomous tractor-trailer may be licensed only to drive on the road, not off road. Its localization algorithm may be more generally licensed for different environments, and its path-planning algorithm may be able to take into account navigating different types of terrain. However, its actual drivetrain system may not accommodate driving on anything other than smooth tarmac. In this case, composition of licensure must carry forward that the suitable environment is a road of smooth tarmac. When one subsystem has been "licensed" (e.g., a target-detection subsystem) and that subsystem is connected to another (e.g., a path-planner subsystem), a key challenge will be characterizing how the licensure elements of the two subsystems work together. Also, how can one specify that the licensure requirements for two subsystems "match" in the necessary ways?

The final step will be to describe these composable licensure requirements over the expected life cycle of the autonomous system. As shown in Figure C-9, the dependency mapping defines the interfaces and interdependencies between operational components of

the system. If appropriately defined, automated formal methods can be applied to the dependency map to ensure that inconsistencies between elements do not combine to create unintended detrimental consequences. In addition, due to the adaptation capabilities of the system and its subsystems, it is expected that such mappings (see Figure C-2) may also need to evolve over time.

Operational Dependency Mapping



Operational Component	Input Requirements	Output products
Detect Stop Sign	--	P_D > 99.999 @ 50m range
Localize Stop Sign	Detection 99.999 @ 40m range Own-vehicle localization (error < $1m^2$)	Stop sign position (error < 1 m^3) @ 50m range
Decide to Stop	Stop sign position(error < 2 m^3) @ 50m range Car detection 99.999 @ 40m range	--
Detect Car	--	Car detection 99.999 @ 200m range
Track Car	Car detection 99.999 @ 150m range	Track prediction error < 5m/sec
ID through traffic	Car detection 99.999 @ 190m range Track prediction error < 5m/sec	Car traffic constraints, P(100%)
ID stopped traffic	Track prediction error < 5m/sec	Car traffic constraints, P(100%)
Decide to travel through intersection	Car traffic constraints = nil, P(100%) Track prediction error < 5m/sec	--

Figure C-9. The final product of the requirements process is a process model that illustrates the relationships between operational components and a dependency matrix that enumerates the required quality of inputs and the produced quality of outputs. By explicitly enumerating component dependencies, we may produce assurance traceability from disparate sources.

Conclusion

This appendix gives a high-level overview of a conceptual process for generating licensure requirements. It represents the first step toward developing an effective overarching process for specifying requirements for licensure.

Many research challenges remain. A comprehensive process for eliciting and articulating thorough licensure requirements will require addressing many R&D challenges, including developing:

- A common, shared vocabulary and expressive language for licensure requirements.
- Methods to describe cross-dependencies of component licensure requirements and when and how they are compatible and composable.
- Methods to show that the licensure requirements are complete enough they will allow composing assurances of suitability for granting licensure under new conditions.
- Template licensure requirement architectures for representative autonomous system use cases, to allow widespread application of this methodology.

Effective requirements elicitation and capture will be a key enabler for evidence-based licensure of autonomous systems.

References

Alberts, D. S,. and R. E. Hayes. 2003. "Power to the Edge: Command… Control... in the Information Age." Washington, DC: Office of the Assistant Secretary of Defense, Command and Control Research Program (CCRP).

Clough, B. T. 2002. "Metrics, Schematics! How the Heck Do You Determine a UAV's Autonomy Anyway?" *Proceedings of the Performance Metrics for Intelligent Systems Workshop* (PerMIS '02).

Defense Science Board. 2012. "Task Force Report: The Role of Autonomy in DoD Systems." www.acq.osd.mil/dsb/reports/AutonomyReport.pdf.

Gillespie, T., and R. West. 2010. "Requirements for Autonomous Unmanned Air Systems Set by Legal Issues." *International Command and Control Journal* 4(2).

Huang, H.-M. 2008. *Autonomy Levels for Unmanned Systems* (*ALFUS*) *Framework, Volume I: Terminology Version 2.0.* Gaithersburg, MD: National Institute of Standards and Technology (NIST).

Kass, Richard A. 2006. *The Logic of Warfighting Experiments.* Washington, DC: Assistant Secretary of Defense (C3I/Command Control Research Program).

[TEVV 2015] Department of Defense Research & Engineering Autonomy Community of Interest (COI) Test and Evaluation, Verification and Validation (TEVV) Working

Group Technology Investment Strategy. 2015. www.defenseinnovationmarketplace.mil/resources/OSD_ATEVV_STRAT_DIST_A_SIGNED.pdf.

Torens, Christoph, and Florian-Michael Adolf. 2014. “V&V of Automated Mission Planning for Unmanned Rotorcraft.” NATO SCI-274 Workshop on Verification and Validation of Autonomous Systems.

Appendix D.
Normative Oracle Generation

Introduction

Given a set of explicit dependability (i.e., performance and assurance) requirements, the licensure framework we have proposed requires implementing normative oracles. The most basic oracles are those that (at a minimum) score the degree to which the observed system behavior meets each fundamental design-independent requirement. Additional normative oracles will be needed for derived design-dependent requirements defined at lower levels of system behavior, including internal behaviors. Such "introspective" oracles, capable of comparing the internal state of the system with expectations of what it should be like and how it should evolve over time, may be essential for licensure of adaptive and learning systems. For example, we know that the numerical weights in machine-learning algorithms should converge over time. If an autonomous system's learning algorithm exhibits oscillatory or cyclic behavior, a normative oracle for learning convergence would score that as undependable behavior, even if the system has been making reasonable choices thus far.

Given the large number of such requirements that are likely to be needed (see Appendix C), automated support for generating these oracles and instantiating them in software is also a high-value area of research and development.

Requirements and Oracles

There is a natural complementarity between requirements and normative oracles. Formal specifications for system requirements lead naturally to oracle specifications. Model-based verification checkers can also serve as normative oracles and potentially as run-time monitors. They can also be used to generate test suites for empirical testing.

One implication of this complementarity is to impose an additional burden on how requirements are stated. There is a considerable literature on testability of requirements specifications[1] and "design for testability."[2] In an evidence-based licensure (EBL) context, there is an added layer of concern, which we will call "design for licensure." Where design

[1] See, e.g., Mark W. Alford, "A Requirements Engineering Methodology for Real-Time Processing Requirements," *IEEE Transactions on Software Engineering* 3.1 (Jan 1977): 60–69.

[2] See, e.g., Harald P. E. Vranken, Marc F. Witteman, and Ronald C. van Wuijtswinkel, "Design for Testability in Hardware-Software Systems," *IEEE Design & Test of Computers* 13, no. 3 (Fall): 79–87, 1996.

for testability involves designing systems with an eye toward being able to test them effectively, and testability of requirements involves specifying requirements in ways that can be easily implemented with tests, design for licensure refers to an overall development approach that is specifically tailored to amass the kind of evidence during development and testing that will support an eventual licensure decision. One piece of design for licensure is making explicit the dense and clearly defined set of requirements that can generate the needed set of normative oracles for the system.

Oracles and Evidence

If the purpose of normative oracles is to amass evidence toward licensure, we need to consider what kinds of evidence are useful beyond simple successful performance testing. Several distinct categories of evidence are relevant to licensure:

1. Evidence from design.
2. Evidence from historical consistency.
3. Evidence of successful corrective action.
4. Evidence from demonstrated robustness.

Evidence from Design

Evidence from design generally involves the use of formal methods (see Appendix B) to produce systems with provable properties. The corresponding normative oracle is equivalent to verification that the formal method has been correctly applied. Unlike most normative oracles, this might be implemented as quality-assurance processes during development and manufacturing, rather than via observation of the working system.

Evidence from Historical Consistency

Systems that exhibit constant or steadily improving adherence to requirements may be deemed more dependable than systems that do not. Normative oracles provide a means to track the "reliability growth" of a system at a level of detail not normally available. A long arc of increasing dependability over the history of development, testing, and deployment justifies more confidence than simply passing a suite of qualification tests at the end of the development process. This is especially true when the observed trend of increasing dependability has been seen in both high-level behaviors and with respect to low-level derived requirements.

Evidence of Successful Corrective Action

Test-diagnose-fix-retest is a standard iterative process within most development projects. From the point of view of licensure, evaluators will have more confidence in a

system if the fixes that have been implemented during development actually tend to solve the problem that was diagnosed. This indicates, among other things, that the developers' mental model of cause and effect within the system is accurate—that the system behaves the ways it does for reasons that are understood. Conversely, when successive attempts to fix a problem fail to change the unwanted behavior or introduce new problems, this calls into question the degree to which the designers understand their own system.

When an implemented fix causes the associated oracles to agree that the problem is not happening anymore and does not cause other oracles to begin to complain, that not only looks like progress, it looks like a process that is in control. If this happens repeatedly, or with increasing frequency over the course of development, that history of stable improvement is evidence of dependability.

Evidence from Demonstrated Robustness

One hallmark of autonomous systems is that they are expected to operate in novel situations. The designers do not assume that the operational state space for the system can be enumerated or exhaustively tested. As a result, when oracles report acceptable behavior even when the system is exercised outside its past environmental and/or mission envelope, this robustness is evidence of dependability. This is particularly important for systems that adapt and/or learn over time—graceful response to unexpected or unintended circumstances is much stronger evidence of dependability than successful accomplishment of a planned mission scenario. For learning systems, it will be useful to preserve as much of the normative oracle instrumentation as is feasible even after the system has been fielded, to either accumulate further evidence of dependability in a broader range of mission contexts or to diagnose unanticipated fragility in the system.

Different Normative Oracles for Different Levels of Behavior

Because the behaviors that normative oracles will assess run the gamut from low-level design-specific behaviors (e.g., acceptable voltage ranges on specific circuits) to high-level mission-success criteria (e.g., landing safely), there is a corresponding wide range of methods for implementing these oracles. At the lowest levels, this will generally look much like typical test instrumentation—additional hardware and software to monitor operating status during developmental testing in ways that would not (and often could not) be monitored during actual fielded system operation. These oracles will generally be highly design-specific and will reflect the engineers' beliefs about why this design works. The challenge at this level is to make the oracles as nondisruptive as possible, so that the overall behavior of the system is not affected by the presence of the oracles.

Higher level oracles cannot be based only on adherence to design specifications. Instead, they must apply design-independent criteria associated with dependable operation. The highest levels of requirement—those that translate directly into safe operation and

mission success—will generally be straightforward to formulate and assess. It will also be necessary, though, to both formulate and assess derived behavioral criteria that are neither design specifications nor mission-performance requirements. This intermediate level of behaviors will in general be both the most difficult to instantiate in a comprehensive set of normative oracles and the most useful for amassing evidence toward licensure.

As an example, consider a driverless car intended for use on public roads. The highest level dependability requirements involve obeying traffic laws, not causing or suffering accidents, etc. Specifying these requirements and describing normative oracles for them are straightforward. At the same time, the lowest level derived requirements will be design-specific assertions about the operation of the cyber-physical system and its algorithms. Oracles for these behaviors may also be straightforward to codify for the engineers who designed them.

In the middle ground, however, are subtle issues associated with what "good driving" looks like. Does the vehicle hesitate too long before pulling out from a stop sign or green light? Does it choose paths that use too many left turns at busy times of day? Does it accelerate and decelerate in ways that makes human passengers uncomfortable? These are all legitimate questions of dependability, but are much harder to formulate as normative oracles or to evaluate quantitatively. They cannot be "instrumented" locally, but require some degree of global assessment. They may involve subjective judgments by human experts. They are design dependent, but address questions of whether the system is performing as needed, as opposed to performing as designed. Nevertheless, normative oracles for these behaviors would be invaluable (and might be essential) to support eventual licensure of such a vehicle.

Implementing Normative Oracles

For low-level oracles, current developmental test best practices are almost enough to support EBL. The process of characterizing desired behavior, instrumenting to detect deviations from desired behavior, and diagnosing deviations from desired behavior is well understood. From an EBL point of view, the only new part is establishing a time series of dependability growth and making sure that the instrumentation supporting an oracle is only removed after the case for dependable performance against that oracle has already been made.

Similarly, for high-level, design-independent requirements, the associated normative oracles will be invoked as part of the usual qualification and operational test and evaluation process. As with low-level oracles, the novel aspect here is the establishment (from an earlier phase of development than is currently typical) of a time series of such assessments. The normative oracle provides a stable standard of performance that allows the system to demonstrate stable convergence to dependable behavior over the long arc of development and into fielding.

As noted above, the most significant changes from current practice would be associated with the middle tier of normative oracles. Unlike the lower tier, these oracles would not assess whether the design has been implemented correctly, but instead whether it is the correct design to generate the desired behavior. Unlike the higher tier, the behaviors to be assessed are not design independent, and thus do not flow transparently from the mission requirements. There will generally be enough of these mid-tier oracles that instantiating them in hardware and software for automated evaluation will be useful, perhaps even necessary. This will require a new form of "instrumentation" that watches for behaviors that may not be easy to define in terms of system states. As a result, designers and developers will need to think in unfamiliar ways about what dependable performance looks like at the subsystem or algorithm level, rather than at the system or component level. Most current system developments feature neither formal specification of normative standards for system behavior at this level nor instrumentation of such behaviors.

Summary

Normative oracles provide a mechanism for establishing confidence in system dependability through a time series of observed adherence to objective standards for desirable system performance. They are defined at the implementation level, the design-adequacy level, and the mission-requirement level, with each level requiring a distinct approach to generating and instantiating oracles. While the highest and lowest levels of requirement are minor extensions of current practice, the middle tier of oracles requires significant changes to how subsystem requirements are derived and how systems are instrumented for developmental test. Taken together, these three levels permit the accumulation of all the relevant types of evidence that can support a licensure decision.

Appendix E.
CoActive Design

Evidence-Based Licensure: Bounding the Confidence Space

Autonomous systems will require human-machine teaming and licensure procedures that explicitly account for it. Such teaming, as discussed in the main paper, may support different operating modes with close control or teleoperation at one end of the spectrum and full machine autonomy at the other. Most approaches focus between the two extremes and mainly on the problem of control and task allocation. A contrasting approach is to focus on interdependence of the human and the machine performing a joint activity. We might consider this an extension of teaming among humans, but that would imply a high degree of machine sentience. One term proposed to describe human-machine teaming is "interdisciplinary coordination," a functional perspective presented in a survey article (Malone and Crowston 1994). *Coactive design*, a more recent term, focuses on the interdependence of the teammates and the design implications that follow from it (Johnson et al. 2011; 2014). This appendix will address four questions that relate coactive design to autonomous systems and evidence-based licensure (EBL):[1]

1. What is coactive design and how is it relevant to autonomy?
2. What support do these methods offer to development of autonomous systems?
3. How can these design methods support EBL of autonomous systems?
4. What are the unique contributions (if any) of coactive design methods to EBL?

Background

The typical approach to designing interactions between humans and machines is to perform an a priori analysis of the tasks/subtasks a job requires and then allocate those to one or the other. This process frequently is thought of as designing the machines to take account of the people who use them. For autonomous agents, Johnson et al. (2014) say this approach focuses mainly on the problems of task allocation and control. For example, the U.S. Air Force (Endsley 2015) proposes a taxonomy of autonomy to extend and complement human performance:

[1] The intent here is to describe how coactive design teaming concepts can apply to the development and test, evaluation, validation, and verification (TEVV) of autonomous systems. The reader interested in the history of these teaming concepts is encouraged to begin with the referenced articles and their extensive bibliographies.

- Implementation aiding—where the system carries out tasks for the human, such as flight management systems or smart weapons that follow human targeting, but the human makes all decisions,
- Situation awareness support—in which disparate data are fused to provide integrated information relevant to [human] operator decisions and goal states,
- Decision aiding—where the system provides a list of potential options [to the operator] and rates or ranks those options as with a recommended target list or course of action …,
- Supervisory control—where the system controls all aspects of a function automatically, including taking in information, deciding on correct actions and carrying out those actions, but the human can set goals and intervene as needed [also called on-the-loop control].

Autonomous agents today mostly support the human, but the reverse can also be true and so can shifting control and task responsibility back and forth. It is useful to view teamwork activities as interdependent and changeable because of collaboration in which task allocation varies over time and situations. Coactive design asks how the team members can best collaborate to accomplish the tasks. Teams have partners who operate jointly, with responsibility and control depending on the circumstances.

There is a long and continuing interest in ways to improve coordination between people and the autonomous capabilities of computers. One approach has been to develop different interfaces such as keyboards, mouse, tracking devices, touch screens, and voice. Another approach is "smarter" software that routinely completes words/phrases and corrects spelling and grammar—though not always accurately. Systems that adapt and learn also provide tailoring to individuals to facilitate performance. For example, an adaptive system for teaching forms a partnership with a student and tailors instructional materials, speed, and details of their presentation based on the student's evolving knowledge and skills. Many of us imagine what forms of coordination in the future are likely to become common.

Future coordination mechanisms for human-machine systems may need to be more flexible than a military command-and-control hierarchy, in which there is limited discretion to choose when and what to communicate. Malone and Crowston (1994) address the need for control structures and processes to coordinate dependencies between people and machines that are emerging in the electronically connected world. They suggest looking for analogies with coordination in existing capabilities. For example, what could we learn about trade-offs that computer systems make between processing/calculations and servicing input-output/communications in distributed systems? How would that illuminate

possibilities for collaboration between humans and computers, and what are ways to manage dependencies between them? Answers require segmenting tasks, sequencing them, sharing information necessary for coordination, and having primary and backup plans among others for accomplishing the tasks. Coordination and collaboration also require processes for managing and representing information flow. Should management of the human's and the machine's resources be done by standardization, first-come-first-serve, scheduling or participatory design, or some other mechanism? In addition, ways to represent and classify the coordination logic are needed by developers and testers. Coordination interdependencies could be outlined with a flow chart or may benefit from more dynamic processes such as state-transition diagrams or Petri nets. There is, of course, no single answer. The control structures and processes must be decided based on an analysis of tasks and efficient ways for the human-machine team to perform them.

Coactive Design and Its Relevance to Autonomy

Coactive design is a concept to address the teaming relationship of humans and increasingly sophisticated agents/robots/autonomous systems. It stresses that autonomous systems, even as they increase in their ability to act independently, will need to work jointly with people not as passive devices but as full partners. Coactive design is the interdisciplinary study of coordination and more specifically the underlying interdependence of participants in joint activity (adapted from Malone and Crowston 1994; Johnson et al. 2011). However, the methods of coactive design and their application are an evolving field explored here in relationship to system design, normative oracles, TEVV and EBL applied to autonomous systems.

Consider an example that characterizes the challenges of coactive design using human-machine teaming to recognize and manipulate objects. A person working together remotely with a robot detects and identifies objects with the help of the robot's sensors before helping the robot to grab and manipulate them using appropriate pressure and care. Humans and machines have relative strengths that can be enhanced by cooperation. A person currently can do pattern recognitions and identify objects more consistently and accurately than a machine programmed to do it. However, the two working interdependently can perform a task sequence—recognizing and manipulating—with more success than either alone. Metrics to assess success are dependent on task/subtask importance, performance requirements, and standards/rules for that performance. What are possible interdependencies (e.g., human can see only by using the machine's optical system)? What is the human's or the machine's capacity to perform under what constraints (e.g., human is remote from the machine's location)? The designers and developers should classify tasks according to their dependencies and how to manage them.

The example serves, of course, only as an introduction to implementing coactive design. Malone and Crowston (1994) encourage us to study the management of shared

resources in a variety of disciplines where it is implicitly or explicitly addressed, such as information technology/computer science, economics and operations research, and organization theory. This foundation will assist designers and developers in deciding how to implement synergistic human-machine teaming, that is, conditions where the human should issue a command and when the automation should override the human and conditions in which the human or machine volunteer information or request information or request physical action. Once a teaming concept is selected, it will be necessary to define normative oracles for that teaming model to allow objective monitoring and quantification of how well the system is achieving its teaming goals, as part of an evidence-based TEVV process leading to autonomy licensure.

Support That These Methods Offer to Autonomous System Development

Current design and development practices do not typically yield formal system requirements that are sufficiently complete, unambiguous, and testable for generation of normative oracles. In contrast, capacities for a human and an autonomous system to support a joint activity are a key element of coactive design methods. They document where one or the other benefits from active support and interdependencies for success. They provide a framework for designing the human-machine interaction and thus identify the kinds of evidence necessary to support the system's viability. A result of coactive design properly done is a representation of how to develop an optimized interactive team.

Coactive design methods help specification developers understand the partners engaged in a joint activity and the particular contributions they could make to support it. The choice of collaboration modes shapes the human concept of operations, the design of the user interface, and the choice of the autonomous system's functional roles in the tasks to be performed. Rather than choosing a collaboration mode a priori, coactive design simultaneously specifies these elements to optimize the efficiency and dependability of the overall collaboration.

As a foundation, Johnson et al. (2014) present a model describing the necessary conditions for successful coactive design: observability, predictability, and directability. That is, the joint activities and signals about them must be observable and interpretable to team members. And the behaviors must be predictable or reasonably reliable. In addition, team members must have the ability to direct one another's behavior. Using these principles, the design methods aim to identify the tasks and subtasks for system design and development that the human and the machine have the capacity to support. Next, select the relationships to build into the design. Refinement and iteration are the norm for good designs, of course, and that is no less so for coactive design. Evaluation of how well the design works compared with a standard of how it should work is the province of EBL.

Use of These Design Methods to Support EBL

Coactive design requires recognizing and managing dependencies between activities of the teammates. Dependencies are a particularly important factor. With people, we ask them to work as a team and they may or may not share the workload and depend on one another. Each of us probably has memories of a supposed team in school with one person doing most or all of the work. In contrast, an effective team consists of members with their own specialties who necessarily depend on one another as the efficient, or often the only, way to accomplish the work. Johnson et al. (2014) capture the challenge for human-autonomy teams:

> Effective teamwork intuitively implies coordination of activity, cooperation among participants, and collaboration. However, all these terms are too abstract to give direct guidance to human-machine systems design and developers. … The challenge is in translating high-level concepts such as teamwork and collaboration into implementation of such concepts within control algorithms, interface elements, and behaviors.

The question is how these concepts and their implementation can be captured in normative oracles to support TEVV and their integration into EBL.

The reciprocal and mutually influencing nature of coactive actions and effects, such as between a driver and vehicle, complement one another. There are required interdependencies such as when a driver engages the accelerator or the brake, and the vehicle responds by starting or stopping. There also are opportunities for interdependence such as when the driver and vehicle use cruise control and trade-off who is the primary and who is the secondary team member, depending on traffic conditions. The car provides certain information (e.g., current speed or engine temperature) on a continuous basis, but other information (e.g., low tire pressure or alternator failure) only in exceptional circumstances. Assumptions about these relationships are incorporated into the vehicle's design, its TEVV, and what can be called evidence for its licensure.

Unique Characteristics of Coactive Design Methods for Contributing to EBL

A coactive design will specify the collaboration protocols between human and machine agents in performing various missions. From this specification, we can derive normative oracles describing appropriate interactive behaviors that the system needs to exhibit in testing. These are the external criteria/standards to capture in oracle-based testing (OBT). Perhaps counterintuitively, these protocols will tend to be most complex in the middle of the spectrum as we progress from teleoperation to near autonomy of the machine agent. Pure teleoperation and pure autonomy involve minimal collaborative interaction; ongoing full collaborative execution of tasks will involve a much richer (and thus harder to validate) information exchange.

What is special about coactive design is that it goes well beyond traditional task analysis. In particular, rather than asking the simplistic question, "Which partner should do this task?" it asks the more subtle question ,"How could the partners best collaborate to do this task?" It has, as other approaches have, an evaluation of team members' informational needs, but also includes knowledge, skills, and abilities such as sensing needs, perception needs, decision needs, and action needs (Johnson et al. 2014), and the corresponding sensing, perception, decision, and action capabilities that can be leveraged by the partnership. This can provide a wide array of protocols for humans and machines to collaborate in performing multiple tasks more reliably. At the same time, defining normative oracles to accurately capture what constitutes "correct collaboration" poses a challenge that is avoided by more traditional and predictable command-and-control protocols.

A well-ordered and -documented design process should greatly help our confidence in licensing a system both initially and after changes are made. Coactive design methods are, however, focused on appropriate *design.* Coactive design leaves implementation to a creative process addressing requirements. It establishes human-machine performance criteria for use in evaluations but offers little guidance about how to do them and no metrics to assess them. It is a starting point without prescriptions or specific implementation methods to accomplish complex human-machine missions using autonomy. Although coactive design makes TEVV harder by increasing the options for human-machine interactions, the achievable performance should be much greater than without it. For EBL, designers need ways to define normative oracles for collaborations that incorporate multiple ways to accomplish the same mission.

References

Endsley, M. R. 2015. "Autonomy Horizons: System Autonomy in the Air Force – A Path to the Future. Volume 1: Human-Autonomy Teaming." AF/ST TR 15-01. U.S. Air Force Office of the Chief Scientist.

Johnson, M., J. M. Bradshaw, P. J. Feltovich, M. Catholijn, C. M. Jonker, B. van Riemsdijk, and M. Sierhuis. 2011. "The Fundamental Principle of Coactive Design: Interdependence Must Shape Autonomy." *Coordination, Organizations, Institutions, and Norms in Agent Systems VI* (COIN 2010 International Workshops, M. De Vos et al., eds.). Springer Link: LNAI 6541: 172–91.

———. 2014. "Coactive Design: Designing Support for Interdependence in Joint Activity." *Journal of Human-Robot Interaction* 3 (1).

Malone, T. W., and K. Crowston. 1994. "The Interdisciplinary Study of Coordination." *ACM Computing Surveys* 26 (1).

Appendix F.
Implications of Learning Autonomous Systems for TEVV

The Challenge

The essential challenge for test, evaluation, validation, and verification (TEVV) posed by autonomous systems that learn is behavior that is not stable and predictable over time. Identical inputs may lead to different outcomes because the system has modified its internal parameters to better satisfy mission objectives. Part of system development must determine what kinds of learning under what circumstances are viable. In turn, licensing generally will include limits on how much learning is allowed and the environments where that learning can be applied. There could be a range of learning levels that qualify for licensing depending on TEVV results. In addition, TEVV must continue after fielding with procedures that focus on monitoring the system and its learning module for indicators of when reexamination is necessary. This appendix will address four questions about the challenges of autonomous systems that learn and relate their implications to TEVV:

1. What are autonomous systems that adapt/learn and their impact on TEVV?
2. Why are changes in TEVV needed as a result of autonomous learning/adaptive systems?
3. How may tools/methods for TEVV be designed for use with these systems?
4. What unique impacts do autonomous learning/adaptive systems have on TEVV?

Background

An autonomous system's design, including any human interactions, is based on the requirement to perform tasks/subtasks that satisfy a mission. For that mission, design/development takes into account the operating environment, inputs to the system of all kinds, and algorithms to control execution. Autonomous systems that learn/adapt will, however, have different internal algorithms and decision capabilities than systems without learning. One without learning will have consistent performance as a function of algorithms whose inputs are sensory data and mission goals and whose execution is controlled/modified by fixed internal parameters. A system that learns will be programmed to modify its internal parameters as a function of experience. So, for example, an autonomous aircraft with the same environmental inputs (e.g., air speed, sensor inputs) and a mission objective to land on a runway may have less than optimal success because its

brakes overheat. As a remedy, the aircraft could change its landing parameters next time to brake more gradually, use more runway for landing, reduce its approach airspeed, and so on. Put differently, the autonomous system that learns can change its internal decision parameters for current and future missions.

During development, both the developer and the system can be making changes to cause significant challenges for performing meaningful TEVV. Initially, the system will have a priori parameters established and inserted by developers to populate algorithms and execute a range of missions. However, the mission and internal parameters for testing performance must change to mimic expected operating conditions. Changes made by developers might be accommodated by systematically generating matrices of values to create a range of operating conditions and system parameters. Generating such values could, of course, be a daunting challenge that may or may not be practical because of the system's complexity. In any case, this challenge exists already for autonomous systems without learning. Such scripted values could be used for testing the system without its learning capability turned on to allow baseline performance measures. Engaging the learning capability that responds to performance experience during development and then fielding is yet another dimension of complexity to evaluate. What kind of impact do these externally and internally driven changes have on TEVV?

Autonomous Systems That Adapt/Learn and Their Impact on TEVV

Autonomous systems that can adapt/learn[1] are those capable of interacting on their own with a dynamic environment to achieve a quasi-structured goal such as landing an airplane or driving a vehicle. This kind of autonomous system begins with core competencies and internal parameters for making decisions. Testing these core competencies with the learning capability decoupled presents the same TEVV challenges as any complex system. Comprehensive testing may be possible in principle but is not necessarily practical. However, the system that learns cannot be comprehensively tested because it responds to environmental inputs over time without prescriptive instructions about correct performance. An interesting challenge for validation testing is that the inputs themselves cause adaptation to occur. Those adaptations may help ready the system for fielding, or they may not provide enough variation for the system to survive an extreme event in the field. The system's development and TEVV incorporate the power of machine-learning algorithms to adjust it—the give-and-take of design and emergence (Tanz 2016).

[1] Autonomous systems that adapt/learn are related, in the academic discipline of machine learning, to a category defined as reinforcement learning in which a "teacher" does not explicitly tell a computer program whether it has come close to its goal (see Ron Kohavi and Foster Provost, "Glossary of Terms," *Machine Learning* 30 [1998]). In this report, autonomous systems can be independent of or interactive with a human as a feature of the environment.

The learning capability depends on algorithms that make predictions from data for deciding what to do next. These predictions cause changes in parameters for decisions that new data can change further. The data may include the actions of intelligent adversaries. An adversary could, in principle, present situations that "teach" the autonomous system during times when little is at stake to behave in an exploitable way when a lot is at stake. The operating environment that causes learning thus is multidimensional. We get an educated guess (Tanz 2016) about what this may be like: "In the future, we won't concern ourselves as much with the underlying sources of [system] behavior; we'll learn to focus on the behavior itself. The code will become less important than the data … to train it."

To assess the system's capability to handle change, TEVV necessarily must incorporate two capabilities: (1) extensive sampling of possibilities during development that stress the learning capability, and (2) quantitative verification of runtime software, and continued system monitoring and periodic recertification of learning systems after fielding. Testing also must assess the system's ability to counter threats from adversaries, including exploitation of its learning capabilities. These assessments represent major changes to the conventional TEVV process and are a topic of particular interest in the software industry.

TEVV Changes Needed as a Result of Autonomous Systems That Learn/Adapt

Deterministic systems that operate in well-defined environments, such as robots on an assembly line, may be complex to test. However, the operating conditions are knowable so that rigorous though not exhaustive TEVV is possible. In contrast, autonomous systems that feature self-adaptive/learning capabilities are not entirely knowable. There is a growing literature to characterize such systems and how to test them represented here by ideas from two papers (Nguyen et al. 2009; Calinescu et al. 2012).

Nguyen et al. (2009) address the test of an autonomous agent with its own internal goals and knowledge, both of which may change over time. Such systems have behavior logic that can differ from an observer's concrete expectations. The tester of a non-learning system may also, of course, not know the algorithms being used or the exact data output from them but can expect the same outputs to given inputs over time. To determine the system's capability over a range of contexts and acceptable outputs, Nguyen et al. (2009) advocate using an evolutionary testing approach guided by a stakeholder's quality criteria that are similar to the normative oracle advocated by the main report this appendix supplements. Evolutionary testing is a term that they and others describe as inspired by classical biology theory with its emphasis on natural selection, inheritance, and variability. Evolutionary testing aims "to evaluate the exhibited performance of the autonomous agents, not the mechanism underlying autonomy itself." The timeline for system adaptation for defense purposes is, of course, far faster than the one from classical biology. The question is whether such autonomous agents are dependable.

Calinescu et al. (2012) reinforce the need for systems that adjust readily. They say a requirement becoming common is that "software must adapt continuously, to respond to changes in application objectives and in the environment in which it is embedded." They also say that software increasingly is expected to fulfill a dependability requirement, a practice that traditionally uses off-line modeling and analysis techniques. Calinescu et al. (2012) advocate bringing the techniques for the two requirements together at runtime to achieve dependable and adaptive software that runs automatically. The capability must be incorporated during development because of high uncertainty about the environment's behavior and its high variability once the application is operational. This approach allows for continuing testing/verification of software to determine if it still meets requirements as it evolves. (We note that continuing testing of the software goes beyond the focus on exhibited performance discussed above. We believe assessment of the decision-making (software) as well as decisions made will be essential.)

The autonomous system needs some set of boundaries to ensure that adaptive behavior is dependable within prescribed limits. Determining those limits and how to test them is a major challenge for TEVV procedures. TEVV has to support, at some acceptable levels of probability, that the system performs reliably and that violations approaching limits of acceptable levels will trigger a recall. The kinds of TEVV changes supported by Nguyen et al. (2009) and Calinescu et al. (2012) as summarized in this section should be considered for regular use with autonomous systems that learn/adapt.

Design of Suitable Tools/Methods for TEVV

The purpose of TEVV is to ensure that the software and systems as developed have a low probability of failing and can adjust to change within acceptable bounds. V&V for well-defined systems can rely on formal methods (see also Appendix B) such as model checking, and proof that theorems, algorithms, and system performance satisfy requirements and specifications. However, systems whose performance is context dependent and that adapt need validation and verification (V&V) that is suitable both before and after deployment and runtime. When and how to test system adaptation properties is a major challenge, albeit with some progress being made.

Morales et al. (2010) advocate norms as a mechanism for coordinating autonomous systems by using explicit obligations, prohibitions, permissions, and associated mechanisms to support self-regulation. These norms are synthesized from prior experiences using a form of unsupervised case-based reasoning. The logic is that a system can generate a solution to avoid the transition to an undesired state based on boundary conditions established and tested during development. Over time, the system evaluates and refines the norms to learn from their cumulative effectiveness. Such a normative system assumes in deployment that similar problems have similar solutions and that undesired states are identifiable.

Another approach focused on self-adaptive software systems (Tamura et al., 2013) comes closer to certifiable V&V for systems that operate in highly dynamic environments, although more work along these lines is needed. They propose explicitly including V&V operations in feedback loops for successful software self-adaptation. Such V&V methods use viability zones characterized as system states that do not compromise operations and, importantly, can change with context changes. V&V aims to keep the system within the viability zones even as they change during deployment. The approach incorporates two levels, each of which require V&V. One level is the target system that dynamically adapts and the other is the adaptation mechanism. The target system has requirements to fulfill, one of which is to satisfy V&V tasks. Another requirement is to adapt to context changes. These must satisfy the adaptation mechanism that also must incorporate V&V tasks. In combination, the target system, the adaptation mechanism, and their V&V provide for the continued and effective operation under varying context conditions.

Both of the above approaches seem to be converging on the type of norm called normative oracles in the main report. Normative oracles are top-down models of desired system behavior in various circumstances to support TEVV of autonomous and adaptive systems with their missions and contexts/environments. Morales et al. (2010) seek ways to control autonomous systems; normative oracles set standards for what performance systems need to satisfy and how well. Tamura et al. (2013) describe explicit V&V runtime tasks to be performed as part of the adaptation process. Normative oracles need such tools and methods for satisfying TEVV of autonomous systems that learn and adapt.

Unique Impacts of Autonomous Learning/Adaptive Systems on TEVV

Traditional computer programming uses explicit step-by-step instructions and may use exhaustive verification approaches, particularly when incorrect processes/decisions have unacceptable consequences. A system that learns can change what it does so that the people who create the programmed instructions never know precisely how the computer accomplishes its tasks. Tanz (2016) reminds us that “machine learning powers large swaths of our online activity” where exactly how the algorithms work is indeterminate. As examples, he mentions that Facebook uses such learning to determine stories in an individual’s news feed, Google Photos to identify faces, and language translators to convert speech in real time. The impact on TEVV is significant.

Menzies and Pecheur (2005), in a chapter about verification and validation and artificial intelligence (AI), focus on the features of adaptive or AI systems (e.g., nondeterministic adaptive knowledge-level systems) that distinguish them from conventional procedural software. The material that follows, based on their insightful review, focuses on the uncertainty of future behavior due to nondeterminism defined as external or internal:

- External nondeterminism results from input or events coming from the environment. Examples include system configuration and initialization, invocation parameters, messages, discrete events, continuous data streams.
- Internal nondeterminism results from the system itself. A common source is concurrency, where scheduling choices are made between concurrent executions (for example, a knowledge system that processes knowledge updates concurrently).

External nondeterminism is easier to test in principle because it is controllable. However, the normative oracle and the operational profile of the system may have a wide range of input possibilities to sample and test. In addition, the tests can become very elaborate. Internal nondeterminism, of particular interest, occurs when a system changes its behavior as a result of new experiences due to data from the environment.

One means to do V&V of an adaptive system is to constrain it to allow predictions of behavior. Menzies and Pecheur (2005) say that a typical way to identify constraints is to instrument the system so that internal choices become visible and can be used in control models. They emphasize that adaptive systems have the benefit of finding ways to fix themselves and derive new behaviors. However, the same adaptation can bring any preadaptation licensing into question. Their ideas about what to do, described next, will not suit every system but do a good job of characterizing the challenges.

What is needed are adaptive V&V criteria and principles to support them. Menzies and Penchur (2005) propose five. They call the first "external validity," aimed at testing how well the adaptation generates models that have some useful future validity. How well does a model work when it is tested with data not seen during training? Another criterion called "learning rates" is how the learning changes over time as more data are processed. How well does it adapt as more data are supplied? "Data anomaly detectors" can be established to pre-filter inputs that are too anomalous according to pre-established criteria and that could, for example, affect the system's "stability" and cause outputs that are unacceptably different. In other words, how does a V&V analyst or a monitor for the normative oracle assess results of the learning? A final criterion can be the "readability" or clarity of the output to an observer (human or automated) from a system that learns. The purpose of each of these V&V criteria is to provide different perspectives and insights into system performance.

Conclusion

V&V methods and tools for systems that learn/adapt are multidimensional, with issues such as how well the system works now, looks ahead, handles different data rates, and so on. The stability required will depend greatly on its purpose instead of on some standard specification. Such criteria are about how the system behaves and adjusts to

performing its mission. Overall, the uncertainty of future behavior requires adaptability in the V&V criteria that must be selected to be system specific. The unique impact of adaptive systems on TEVV is that analysts—human combined with models/algorithms—who thoroughly understand behaviors expected of the system must generate adequate criteria to license, as well as recall it for updates.

References

Calinescu, Radu, Carlo Ghezzi, Marta Kwiatkowska, and Raaela Mirandola. 2012. "Self-Adaptive Software Needs Quantitative Verification at Runtime." *Communications of the ACM* 55 (9): 69–77.

Menzies, Tim, and Charles Pecheur. 2005. "Verification and Validation and Artificial Intelligence." 65: 163–200.

Morales, Javier, Maite L´Opez-S´Anchez, Juan A. Rodriguez-Aguilar, Wamberto Vasconcelos, and Michael Wooldridge. 2010. "On-line Automated Synthesis of Compact Normative Systems." *ACM Transactions on Autonomous and Adaptive Systems* 10, no. 1, Article 2.

Nguyen, Cu D., Simon Miles, Anna Perini, Paolo Tonella, Mark Harman, and Michael Luck. 2009. "Evolutionary Testing of Autonomous Software Agents." *Proc. of 8th Int. Conf. on Autonomous Agents and Multiagent Systems* (AAMAS 2009, Decker, Sichman, Sierra and Castelfranchi, eds.), May, 10–15, Budapest, Hungary.

Tanz, Jason. 2016. "The Rise of Artificial Intelligence and the End of Code. Soon We Won't Program Computers. We'll Train Them like Dogs." *Wired*, May 17. http://www.wired.com/2016/05/the-end-of-code.

Appendix G.
Modeling and Simulation Considerations for Licensure of Autonomous Systems

Don Davis and Don Strausberger

This section addresses modeling and simulation's (M&S) role in licensure of autonomous systems. To do so, key drivers from evidence-based licensure (EBL) and autonomy are first identified to form a foundation for analysis. Both EBL and autonomy are emerging disciplines without well-established taxonomies or descriptions. Second, key M&S components are identified in the context of EBL in the main paper to clearly define the capabilities required. Third, the effects of these three disciplines on one another must be analyzed. Autonomous capability in systems and an EBL approach will both impose constraints on supporting M&S. In addition, there may be areas where M&S capabilities are uniquely suited to supporting EBL. Specifically, the effects of EBL and autonomy drivers on the M&S components need evaluation to help answer the following questions:

1. What differentiates M&S in support of EBL and autonomy?
2. What are the key characteristics of the M&S components required to support EBL of autonomous systems?
3. What role can M&S serve in support of the EBL framework?

Background

Traditional manned systems are operated by humans based on sensor input and an understanding of how the system will respond. A modern fly-by-wire system embodies a lot of autonomy and cannot be said to be truly "transparent," but the behavior is comprehensible to the pilot. In contrast, some autonomous systems are influencing or even replacing human decision-making with algorithms that are not transparent or comprehensible. This imposes additional challenges on the human reasoning and use of judgment when the basis for the systems actions are not understood. This lack of understanding may manifest itself as perceived risk in relying on the autonomous system by an operator and also in a cautious approach to licensure of the system. Sufficient transparency into autonomous system decision-making to support both operation and licensure are a fundamental challenge for EBL.

As introduced in Sections 1 and 2 of the main report, the EBL approach, as a form of test, evaluation, verification, and validation (TEVV), has been embraced by the medical

community and extensively researched for its applicability to "software-based" systems. The National Research Council's study examined software-system dependability over a 5-year period (2004–2009), which culminated in a 160 page report. EBL is unlike other procedural/process-driven approaches such as Hazard Fault Analysis Assessment, Stage Gate Development Process, Kepner-Tregnoe Problem Analysis, or the myriad approaches established to solve particular problems that span multiple dimensions/stakeholders. That is because there are no explicit procedures or steps to EBL. However, there are fundamental characteristics of EBL that can be applied to generate the framework described in Section 2 of the report. EBL is based on dependability described as: "A system is dependable when it can be depended upon to produce the consequences for which it was designed, and no adverse effects, in its intended environment" [1]. The fundamental aspects implicit in the EBL description that affect M&S are referred to here as "EBL drivers."

Multiple sets of autonomy terminology have been generated by robotics, human factors, artificial intelligence. Several of these terminology sets are shown in Figure G-1 in connection with the observe, orient, decide, and act (OODA) loop. Most relevant to this effort is an acknowledgment that multiple taxonomies exist and that none are right or wrong or more or less relevant in the context of EBL. For consistency, the remainder of this section will use the framework established by Endsley and Garland [3]. Endsley distinctly separates situational awareness and decision-making and further decomposes levels of situational awareness into *perception* (recognition of key information and events), *comprehension* (combining, interpreting, storing, and retaining relevant information), and *projection* (abilities to forecast future events and dynamics).

Figure G-1. Common Taxonomies Describing Autonomy

M&S in Support of EBL and Autonomy

The most critical EBL driver for "producing the consequences for which it was designed" is to define behavioral-level requirements and not lower-level properties/specifications of the underlying software or system. Similarity, "and no adverse effects" implies that defining expected behaviors is equal in importance to those that a system *shall not* exhibit. "Behavior" in this context must be understood to include how courses of action are chosen, as well as whether or not they are successful. This needs to be viewed from both the "must" and the "shall not" perspectives. Applications of M&S to expand the examined portion or the state space are common. Applications of M&S to assessing the quality of the decision-making—as opposed to the success of the decision—needs additional exploration.

Licenses in general are not open ended. They need to be bounded with restrictions, require renewals, and sometimes have planned reductions of the restrictions with experience. "In its Intended Environment" implies that the EBL driver depends upon clearly stating any limits on the environment in which licensure is applicable. M&S has the potential to support interpolation and under some circumstances extrapolation of the allowed environment beyond what could be tested.

Ultimately, EBL places emphasis on "evidence," which highlights three characteristics distinguishing it from other V&V approaches:

- The continuous/incremental/temporal gathering of evidence over the product/system life cycle.
- The relevance of analysis (in addition to testing) in supporting the evidence chain.
- The capability of M&S to encompass expectations in addition to actuals as part of the evidence trail.

EBL implies:

- **Desired behavioral-level requirements**
- **Unacceptable behavioral-level requirements**
- **Attention to decision-making in addition to decision outcomes**
- **Defining and bounding environments**
- **Evidence based on both analysis and evaluation, and**
- **Support of continuous evidence gathering and assessment**

As with any evaluation, it is important to clearly define the system under test (SUT). In this case, we need to bound the autonomous subsystems from the rest of the system to appropriately focus on (and define the scope of) the testing of the autonomous features. There are circumstances in which the "situational awareness" elements of the system will

include “autonomous” decisions about the implications of sensor data. This is the simplest case of situational awareness as “perception.” In these cases we think M&S will have a role, especially in exploring error-prone or unreliable data or in making a dependability case that erroneous data will not lead to catastrophic failures. The situation for situational awareness extending to comprehension or projection becomes more complex.

The description of the SUT needs to be explicit about the extent to which there are substantive autonomous capabilities underlying the situational awareness and about any learning capabilities. Both will influence how M&S can support development of a dependability case.

A fundamental challenge of evaluating autonomy is the ability to query the decision-making processes of a machine (versus a human). The need to understand why a decision was made will be equally important to the EBL paradigm as what decision was made. The term “observational analysis” is used to describe this “monitoring” ability, and we need to distinguish two facets of it. The first is a “run-time monitoring” capability. This is the system checking itself in real time (including checking on processes used and quality of decisions being made) and perhaps executing various “fail-safe” options. This is part of the system. In addition, the system will need to record the inputs, algorithms chosen, and so on that lead to decisions being made for the purpose of external evaluation, usually via comparison with the portion of the “Normative Oracle” that applies to the decisions. M&S in support of expanded testing of run-time monitoring is relatively well established in practice. M&S in support of evaluations in terms of a normative oracle remains in its infancy.

We should distinguish “adaptation,” knowledge-based learning and behavioral-based learning to emphasize M&S challenges for EBL. Adaptation includes adjustments to the environment based on sensor data. A simple example might be to adjust the fuel-air mixture in an internal-combustion engine, depending on the measured temperature. However, different kinds of adaptation complicate the situation. Knowledge-based learning includes data collection to support an algorithm that optimizes fuel efficiency based on measurements of temperature and fuel consumption. The relation between the fuel-air mixture and temperature would change as experience was gained. Testing of such a system requires both testing at multiple experience levels and also direct testing of the algorithm (rather than the outcome) that adjusted the mixture. Finally, consider behavioral-based learning. In this case, the system reviews all the data on the environment (temperature, humidity, pressure…) and continually refines the algorithms used to optimize fuel efficiency. New relationships between the environment and the system behavior can emerge that are not be revealed in traditional testing. We anticipate a role for M&S in exploring and bounding the environment.

Key Characteristics M&S Required to Support EBL of Autonomous Systems

M&S is an overloaded term that has different meaning to those in science and technology (S&T) research, development, test, and evaluation. For the purposes of this study we baseline our discussion of the utilization of M&S in the T&E of DoD systems. Within this baseline, M&S still implies different nuances for different stakeholders such as constructs (live, virtual, constructive), or modeling the environment (the elements surrounding the SUT), or modeling the SUT itself.

To provide a more concise description of M&S for the current context, the following components are based on those established in [4]:

- **Test Selection Criteria**: The rules/logic/algorithm used to establish the test suite.
  - **Test Case**: A finite set of input and expected output. For a non-deterministic system this may be in the form of a tree or graph.
  - **Test Suite**: A finite set of test cases
- **Oracle:** Fundamentally, an oracle will assess and score observed system behaviors for desirability. To do so, an oracle must be able to (1) specify/generate the preferred/expected performance/behaviors and (2) compare the preferred/expected performance/behaviors with the resulting performance/behaviors of the SUT and provide the assessment/scoring. To further assess M&S in the context of EBL and autonomy is it advantageous to distinctly separate the two functions and define them as follows:
  - **Normative Model:**[1] The normative model specifies/generates the preferred/expected performance/behaviors of the system. It is an essential component of M&S with the following characteristics:
    - Generates higher level output behaviors.
    - Supports the continual gathering of evidence.
    - Supports different levels of past experiences.
    - Supports sufficient granularity in situational awareness and decision-making (i.e., introspection) so that "why" a decision was made can be evaluated by the assessor.
  - **Assessor:** This component compares the expected performance/behavior of the SUT (the model's output) with the actual performance/ behavior of the

[1] The referenced description [4] employs the term "Performance/Behavior Model," which is changed here to conform with the current report's term for autonomy licensure and its TEVV.

SUT and scores the desirability. The desirability may be binary (meet/does not meet), enumerated (no good, good, great), or weighted/scaled ("8.271 out of 10").

- **Environment Model:** The component that generates the environment that the SUT interacts with. Note that the environment would include generation of inputs that the autonomy perceives and would act upon the decision-making of the SUT. Since the SUT is the autonomy, other components such as virtual target generation and vessel/vehicle dynamics all become part of the environmental model.

M&S includes the following key components:

- **Test Selection Criteria**
- **Oracle**
  - **Normative Model**
  - **Assessor**
- **Environment Model**

Impacts of EBL and Autonomy on M&S

To understand the impacts of EBL and autonomy on M&S, each of the EBL and autonomy drivers introduced earlier must be evaluated with respect to the M&S components. This impact assessment is summarized in Table G-1. Column 1 specifies whether the originating source of the driver is EBL or autonomy. Column 2 contains a short description of the drivers. Columns 3 through 6 represent each of the M&S components. Each of these four columns are further analyzed below to determine the key requirements of each M&S component.

Column 3 of Table G-1 provides insight into the test selection criteria component for M&S in support of EBL. The need for test selection criteria to support negative behavioral-level requirements is challenging but not unique to EBL. Simply stated, the criteria used to generate the test cases must be sufficiently broad and granular to produce sufficient evidence that the SUT will *not* misbehave in particular manners at particular times. During traditional T&E, this often leads to an exhaustive input set to generate and evaluate against and is commonly referred to as "state space explosion." Implicit in EBL is the hypothesis that defining behavioral-level requirements (versus lower level properties/specifications of the underlying software) *and* bounding the input environment sufficiently constrains the input/output relationships so that a finite test suite can be defined for EBL. This will be further explored in the use case in the following section.

Environment Model

Column 4 of Table G-1 provides insight into the environment model component for M&S in support of EBL. The environment model must support tests for both positive and negative inputs at the behavioral level and support a method of bounding/constraining the environment. The environment model should be independent of the autonomy implementation. Ideally different instantiations of the autonomy can be evaluated using the same environment model.

Table G-1. EBL Drivers, Autonomy Drivers, and Their Impacts on M&S Components

Driver	Description	M&S Components Impacted			
				Oracle	
		Test Selection Criteria	Environment Model	Perf./Behav. Model	Assessor
EBL Driver	Support Behavioral Level Requirements		X	X	X
	Support Negative Behavioral Level Requirements	X	X		X
	Support Environmental Bounding	X	X		
	Support Continual/Incremental/Temporal Gathering of Evidence			X	
	Analysis to support evidence chain	X	X	X	X
Autonomy Driver	Inclusive of Situational Awareness and Decision-Making			X	
	Independent of autonomy implementation paradigm (BDI, etc.)	X	X		X
	Supportive analysis			X	X
	Supportive of learning	X		X	X

Normative Model

Column 5 of Table G-1 provides insight into the normative model component for M&S in support of EBL. EBL specifically requires the model to expressly support higher level behaviors and not lower level or granular performance expectations. Note also from column 5 that the model must support the continual gathering of evidence as well as support learning-capable systems. This implies that the model would need to support evaluation over the life cycle of the autonomy (which may exhibit a wider breadth of behaviors as it

matures or as the environmental bounds are extended). Supporting learning implies the model must also support different levels of past experiences (one may envision novice, intermediate, and experienced "autonomy").

Assessor

Column 6 of Table G-1 provides insight into the oracle for M&S in support of EBL. The assessor inputs are the SUT output and normative model output (along with supporting introspection information). Particularly relevant to EBL is the evaluation of higher level behaviors (or particular aspects of behaviors) of the autonomy, rather than lower level performance comparisons. Learning presents a unique challenge to the assessor (and overall M&S) because the experiences of the actual SUT and normative model must be accounted for in test design and execution.

Based on the above we derive the following unique and differentiating aspects of M&S components that are required to support EBL of autonomy:

The environment model must support mechanisms to constrain/bound the environment in multiple dimensions to effectively support EBL.

EBL takes a positive step in addressing undesirable output behaviors, but the fundamental challenge of generating comprehensive evidence (such as a "comprehensive" set of test cases) to ensure no undesirable outputs remain.

Beyond operating upon higher level behaviors driven by EBL, a normative model is affected by autonomy drivers and must support varying "experience" levels for autonomy that can learn as well as observable behavior for the assessor.

The assessor component of the oracle must evaluate higher level behaviors in the context of "why" and in the context of behavioral performance outputs that the assessor can observe.

References

[1] D. Jackson, M. Thomas, L. I. Millett, and T. Baker, "Software For Dependable Systems," ed: Wiley Online Library, 2008.

[2] D. R&E, "Autonomy COI ATEVV Working Group Technology Investment Strategy 2015-2018," ed, 2015.

[3] M. R. Endsley and D. J. Garland, *Situation awareness analysis and measurement*: CRC Press, 2000.

[4] M. Utting, B. Legeard, and A. Pretschner, *A taxonomy of model-based testing*: Department of Computer Science, University of Waikato, 2006.

[5] D. Strausberger, “Autonomy T&E Infrastructure Gap Study Phase 1 Final Report,” G. T. R. Institute, ed, 2016.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED *(From–To)*
March 2016	Final	Jan 2016 – Mar 2016

4. TITLE AND SUBTITLE

A Framework for Evidence-Based Licensure of Adaptive Autonomous Systems: Technical Areas

5a. CONTRACT NUMBER
HQ0034-14-D-0001

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Tate, David M.
Grier, Rebecca A.
Martin, Christopher A.
Moses, Franklin L.
Sparrow, David A.
Edmonson, James R.
Chaki, Sagar
Scheidt, David H.
Piatko, Christine D.
Davis, Don
Strausberger, Don

5d. PROJECT NUMBER
AK-2-3944

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Institute for Defense Analyses
4850 Mark Center Drive
Alexandria, VA 22311-1882

8. PERFORMING ORGANIZATION REPORT NUMBER

IDA Paper P-5325
Log: H 16-000680

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Air Force Research Laboratory
2610 7th Street, Bldg 441
Wright-Patterson AFB OH 45433

10. SPONSOR/MONITOR'S ACRONYM(S)

AFRL

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release; distribution is unlimited (1 August 2016).

13. SUPPLEMENTARY NOTES

14. ABSTRACT

Adaptive autonomous systems of interest to DoD have great potential to complement human performance in a wide range of missions. This particularly is true for adaptive systems that learn—those whose behavior on a given set of inputs may change over time, even after the system has been fielded. However, such adaptation makes exhaustive testing, certification, and licensure of the final system impossible. The challenge is to establish high confidence that the system will perform dependably and behave as intended while safely, securely, reliably, and effectively carrying out the assigned missions. These topics are outlined in IDA Paper P-5325, "A Framework for Evidence-Based Licensure of Adaptive Autonomous Systems." This paper, a companion volume to P-5325, provides additional technical detail on six topics: (1) Formal Methods; (2) Requirements and Metrics; (3) Normative Oracle Generation; (4) CoActive Design; (5) Implications of Learning Autonomous Systems for Test, Evaluation, Verification, and Validation; and (6) Modeling and Simulation Considerations for Licensure of Autonomous Systems.

15. SUBJECT TERMS

autonomy, TEV&V, adaptation, Normative Oracles, licensure, certification, dependability cases

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Ms. Kristen Kearns
Uncl.	Uncl.	Uncl.	SAR	62	19b. TELEPHONE NUMBER *(include area code)* (937) 656-9758

Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std. Z39.18